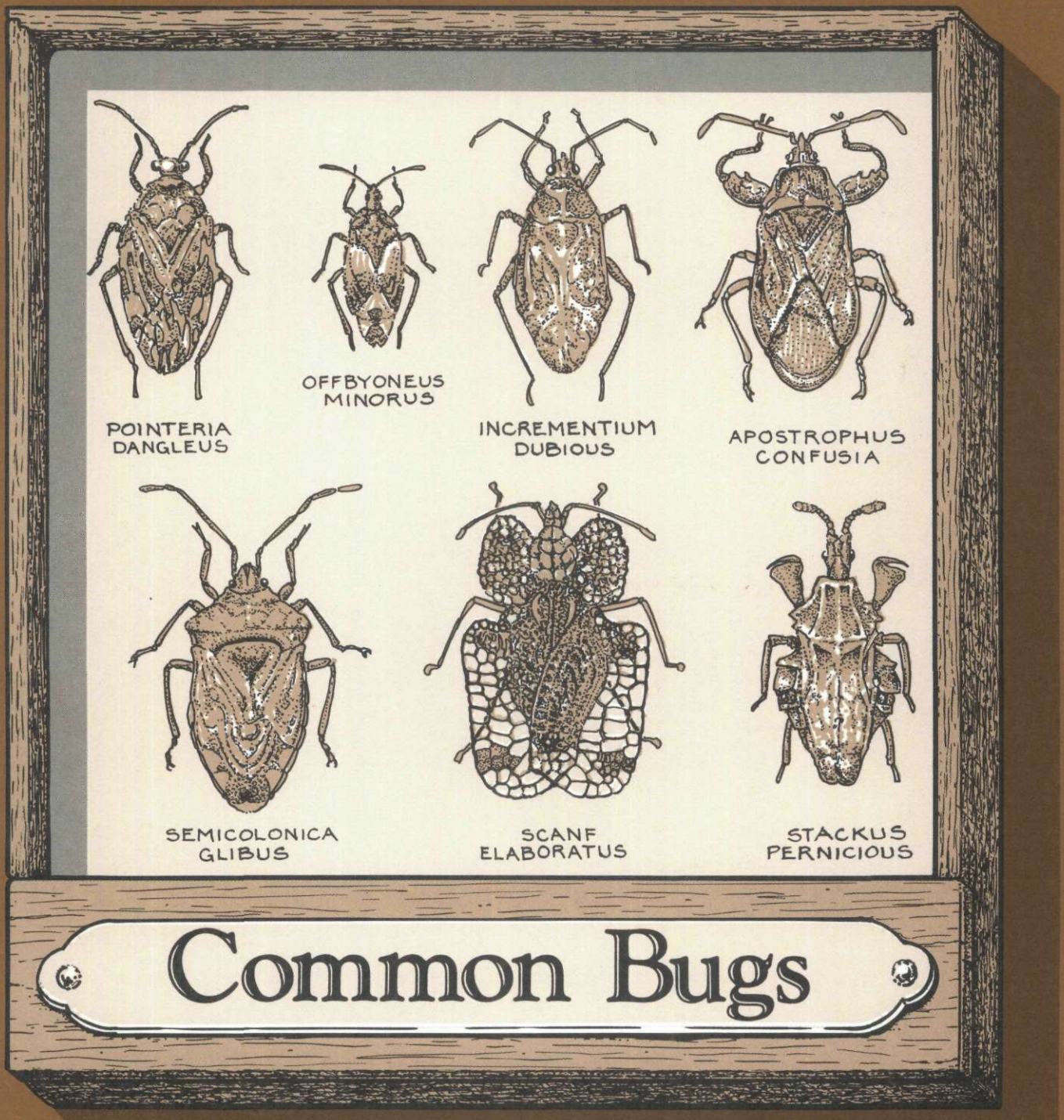
Vol. 3, No. 6 November/December 1985 $4.00

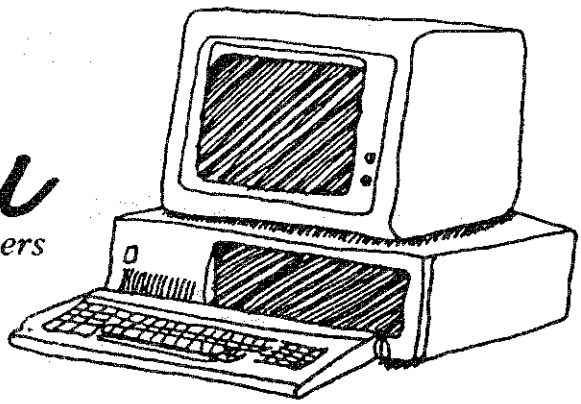
programmer's journal

The Resource Journal for IBM PC Programmers



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal
The Resource Journal for IBM PC Programmers



Volume 3, Number 6 November/December 1985

CONTENTS

HOUSEKEEPING: *THE 3rd ANNUAL PERSONAL COMPUTER FAIRE — SAN FRANCISCO* 2

LETTERS 4

PJ FORUM: C's HLL OVERHEAD by FNE 10
Excerpts from DTACK GROUNDED: The 68000/32081 Systems Journal
Additional remarks by Peter Norton of Norton Utilities

CONSULTING: HOW TO RECOVER ALL YOUR COSTS by Paul Barkley 14
What about those unbillable hours?

REDIRECTIONS: REPORT ON THE COMDEX AND USENIX SHOWS by Frank Greco 18
Two shows as different as their locales

OS: SANTA CRUZ XENIX FOR THE IBM PC by Ron Moak 24
Starting up XENIX

C: COMMON C BUGS by Randall Swan 28
Favorite entomological specimens from an experienced C teacher

PASCAL: TURBO PASCAL 3.0 AND SUBDIRECTORIES by Neil Rubenking 33
Access directory information from within your Pascal program

dBASE II AND III: MORE dBASE II ROUTINES by Brett Salter 38
The second installment of Brett Salter's routines

BASIC: A BETTER WAY TO INTERFACE BASIC AND ASSEMBLER by John W. Jackson 41
Recover memory from a subroutine, and more

REVIEW: CLIPPER SAILS ON dBASE SEAS by Paul Barkley 45
A newcomer puts some wind in those sails

REVIEW: CTOOLS2 by Rex Jaeschke 49
More time-saving routines from Blaise

MISC. 51

PJ BILLBOARD 52

AD INDEX 53

ISSN# 0747-5861

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal

Publisher:	Elizabeth Oakley
Editor:	Robert Keller
Managing Editor:	Erin O'Connor
Production Manager:	Ralph Bentley
Circulation/Promotion:	Robert Tappan
Cover:	Ralph Bentley
Illustrations:	Julia King
Production Assistance:	Holly Tracy

Contributing Writers:

FNE	Ronald Moak
Paul Barkley	Peter Norton
Frank D. Greco	Neil Rubenking
John W. Jackson	Brett Salter
Rex Jaeschke	Randall Swan

Founder: Greg Estes

Business and Editorial Offices:	Programmer's Journal 29 West 29th Avenue Eugene, OR 97405 or P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07843 (503) 484-2162
Postmaster send address changes to:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834

Subscriptions: Bi-monthly rates $24 U.S.
($30 in Canada and $40 elsewhere)
2 years $44 U.S.
($55 in Canada and $75 elsewhere)
3 years $64 U.S.
($80 in Canada and $110 elsewhere)

Advertising: For ad rates, deadlines, and placement, contact Ralph Bentley, (503) 484-2162, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403

IBM and Personal Computer are trademarks of International Business Machines; dBASE II and dBASE III are trademarks of Ashton-Tate.

PROGRAMMER'S JOURNAL (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company. Application to mail at second-class postage rates is pending at Eugene, OR.

© 1985

THE THIRD ANNUAL PERSONAL COMPUTER FAIRE — SAN FRANCISCO

Three members of the *PJ* staff motored down from Oregon for San Francisco's 3rd Personal Computer Faire, which was somewhat better attended than the 2nd Faire. Is the summer computer slump ending?

The Jackintosh was there, running a dull game from Lucasfilm with amazing graphic speed. The Apricot computer line from England was a couple of aisles over: MSX Corp. of Hayward is asking $995 for the Apricot F-1, an 8086 PC-compatible with a single 3½-inch drive, mouse, 512K, color driver, parallel and serial ports, and lots of software. The Apricot is evidently doing poorly in the U.S., although it's hard to see why. A number of Far East XT knock-offs were there for under $950, lead by the VIPC/XT from VIPC Computers in Hayward: $699 for a 256K XT clone with a single drive.

Computer celebrities wandered the aisles. Adam Osborne, whose *An Introduction to Microcomputers* I read until it fell to tatters, manned a simple booth and demoed Paperback Software's VP-Planner. IBM had a big display upstairs. (They also ran a technical session on programming for the TopView environment that had all the levity of a morgue.)

Borland had a spacious display, featuring a large neon lightning bolt that kept causing their software demos to lock up. Philippe Kahn, Borland president, was there, and told of his move from France to Silicon Valley several years ago, armed with a French doctorate in mathematics, a "few thousand dollars," some Apple experience, and a desire to do more with computers.

Before coming, he cabled 100 companies asking for a job interview, and only ten responded. Only one, Hewlett-Packard, gave him much time, but they didn't give him a job.

Philippe had brought the Turbo Pascal compiler with him, so he developed a business plan and "spent a lot of time sitting in lobbies." At an asking price of $50,000, nobody, including DRI and Microsoft, was interested.

One company commissioned a market survey that said that Turbo would sell a maximum of 20,000 copies.

Philippe finally borrowed his father's last $20,000, bought an ad in *Byte*, set up shop in a garage, and sold $150,000 worth in the first month.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

More from Philippe:

On Marketing

"Ours isn't really a horizontal market piece of software, it's more a diagonal market. Turbo costs us $1.20 to produce. It's shrink wrapped and very simple. In June we shipped 146,000 units. There are 400,000 registered copies of Turbo out there, and 400,000 copies of Sidekick.

"I demoed Turbo for Microsoft. No one was interested at all. I took it directly to the user. Users are smart. Don't underestimate them. They want a product that is a useful tool. Apple's marketing is all wrong. They had the fold-out of the first page of *Byte* magazine for life but they gave it up and now we have it.

"They didn't know how to use it. Look what they ran there — a nice picture of a kid's room after school, and the turtle in the aquarium and the Apple IIc. Just the thing to make a parent guilty, and make them think 'I gotta get my kid a computer.'

"That ad should run in *Business Week*. *Byte* has 400,000 readers who are power users and hackers. You need to be specific, to tell the user what he is getting, what this tool will do.

On Software Piracy

"I think the main stimulus to software piracy is the high price of software. Look, you wouldn't expect a car dealer to not let you have a test drive, and to say 'Look, you have to buy these four cars and then throw away the three that you can't use.'

"I think the whole definition of piracy is wrong. According to the definition, I'm probably the biggest pirate here. But if I can use a piece of software, then I'll go out and buy it, just to get the support, and because I think it's useful. My first copy of WordStar, that I got in 1979, was pirated, but since that time I've bought 12, 13 copies. So instead of not buying one copy-protected disk, I've bought a lot of copies.

"I think there has to be some basic trust when you sell somebody software. If the software is copy-protected, you're saying, 'Look, you're a thief — buy this software.'"

—Robert □

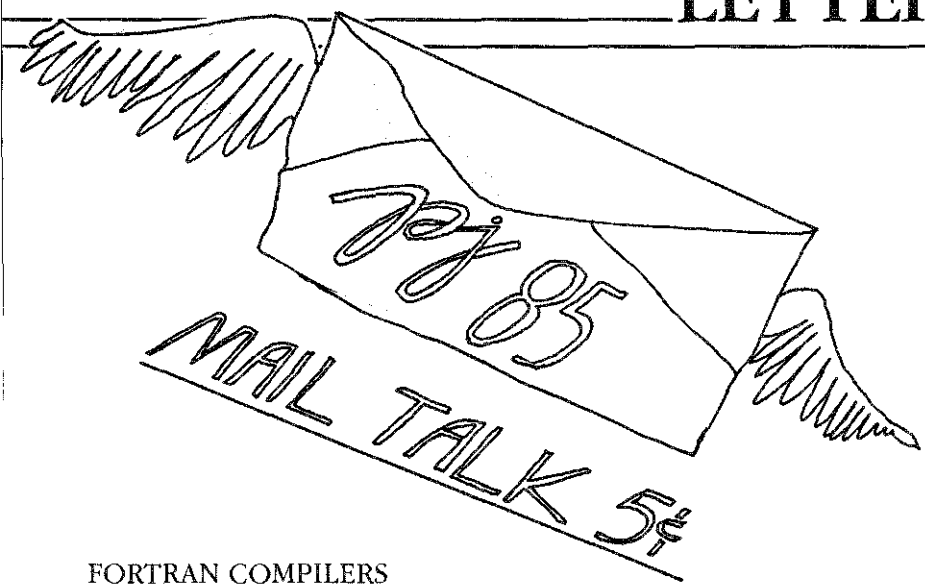
Larger. . . We are moving our offices into a new, larger space. We are also giving *you* more space in our magazine, with more pages in this issue than ever before! More **Programmer's Journal** means more articles from authors like Peter Norton, Tom Swan, and Frank Greco. It means a new, wide open letters section. More of the distinctive graphic art of Julia King and other artists. More information on the latest in software development tools.

Our phone number remains the same; (503) 484-2162. And you can still reach us at P.O. Box 30160, Eugene, OR 97403. **Programmer's Journal** is still brought to you bi-monthly for only $24 per year.

programmer's journal
The Resource Journal for IBM PC Programmers

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS



FORTRAN COMPILERS TESTED

I am truly amazed that you are still in business. You have created an absolutely useful publication of very high and consistent quality. This is a sure-fire formula for failure on television. Other magazines catering to the PC market seem to be written by bozos who've never been paid for a line of code in their lives — they're the Dukes of Hazzard or Love Boat of PC-specific publishing. One expects them to thrive. *PJ* is just so different, so much more significant. I really didn't think you had a chance. And, am I glad I was wrong!

Anyway, that's not why I'm writing. I sold a customer an APT processor for numerical control programming a couple years ago. It was written by another company in MICRODAFT FORTRAN 80 for a BABY BLUE CP/M emulator board. My customer recently wanted to expand, and it seemed like a good time to get into the current native language version of NCAPT.

The vendor was happy to run some benchmarks on his native version to compare with the Z-80 BABY BLUE version, but the results seemed wacky. He was reporting times much greater than normal. It was completely unbelievable, so I ran my own tests. They confirmed the vendor's — both the new MICRODAFT FORTRAN marketed by IBM and the wonderful DR FORTRAN 77 are very very slow.

I am enclosing a copy of my tests. Your readers may be interested in the run times and some of the conclusions.

By the way, you may detect just a little lack of reverence for the Bellevue Bozos in this letter. I think that deep down they really have no idea of what they're doing. Know anyone who programs for a living that would argue the point?

My sincerest wishes for your continued success, and good luck.

RESULTS OF NON-EXHAUSTIVE BENCHMARK TESTING OF FORTRAN COMPILERS

2/24/85 — D. Lippincott

Compilers tested were

1. IBM FORTRAN 2.0
2. DIGITAL RESEARCH FORTRAN 77 4.0
3. MICROSOFT FORTRAN 80. vsn. 3.44 running on a BABY BLUE board in a PC (i.e., a Z-80 based CP/M emulator board for the IBM PC).

Notes:

1. IBM and DR compilers were run on an AT with 18MHz xtal. FORTRAN 80 was compiled on a BABY BLUE CPU PLUS board.
2. Except as noted, all executions were on a standard PC.

Test 1 — Primarily floating point arithmetic. (This is the IBM FORTRAN PACKAGE "DEMO" program.)

parameter	F-80	IBM	DRI
size of .EXE program	12416	29994	84048
compile time (in sec) (see note 1)	6	17	14
link time (in sec)	29	55	114
run time without 8087 simulation	28	10	n/a
with 8087 simulation	n/a	26	339
with real 8087	n/a	5	9

Test 2 — Integer arithmetic. (This is the DIGITAL RESEARCH SIEVE program.)

parameter	F-80	IBM	DRI
size of .EXE program	20096	40144	97664
compile time (in sec) (see note 1)	6	18	15
link time (in sec)	26	19	55
run time	15	27	21

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CONCLUSIONS

1. The DRI product is useless without 8087 if there is any floating point. With an 8087, the IBM product is still much faster. DRI is marginally superior to IBM and inferior to FORTRAN 80 for integer operations. Disk performance would depend on the application.

2. The IBM product is probably best overall, considering limited applicability of BABY BLUE boards, larger address space, etc. Even so, due to its lower integer and disk speeds vis-a-vis the Z-80 product, pride of author(s?)ship is inappropriate.

3. Both programs were small (less than 50 lines). The ratios for code size and link times would probably become closer for larger programs.

4. If FORTRAN is a must, and programmer time is worth more than five dollars an hour, an AT can probably be cost-justified.

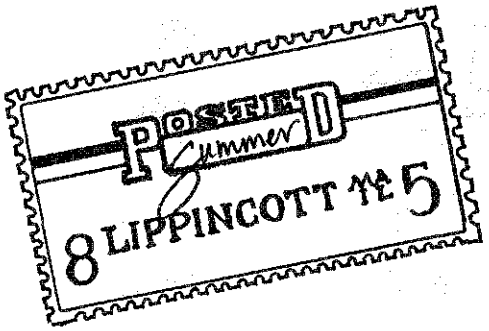
5. A high level of source code compatibility was observed between the DRI and IBM products.

6. The MICROSOFT FORTRAN 80 PRODUCT was not available for testing. It is expected that its performance would be very close to the IBM version's.

7. Compile and link time options are available that were not optimized for the various programs. It is probable that the overall results would not be significantly different had such optimization been attempted.

8. Had compilation been run on a PC, the difference between FORTRAN 80 and the 8088 based products would have been staggering.

Doug Lippincott
Stanton, CA 90680



C PROGRAMMERS CONTROVERSY

In the survey on C programming books (PJ, July/Aug 1985), Frank D. Greco claims to find a number of errors in the C books, but in most cases either the book is correct and Mr. Greco is wrong, or the "error" is a minor typo or slightly imprecise statement that would mislead only the most naive reader. Here are some of the so-called errors that Mr. Greco finds:

Mr. Greco claims that

```
int x;
x = ++x;
```

is not portable, and suggests

```
int x, y;
y = ++x;
x = y;
```

as a portable substitute. In fact, x = ++x will produce the intended effect on all C compilers although it is preferable to say

```
int x;
++x;
```

Mr. Greco says that the statement "The definition of a . . . structure, union, enumerated data type, or typedef does not cause storage to be allocated" is wrong. In fact, the statement is completely correct. Defining data allocates storage, but defining a data type does not.

Mr. Greco wonders why "so many authors confuse the distinction between a definition and a declaration, when K&R make the difference very clear." In my opinion, this distinction is the LEAST clear thing in K&R. [K&R is the book by Kernighan and Ritchie on The C Programming Language, also known as the C Bible.] K&R says that data declarations outside function definitions are assumed to have the storage class specifier "extern" if none is given. If one takes this literally, then

```
int x;
```

and

```
extern int x;
```

should have the same meaning outside a function. If so, then it is not clear whether such a declaration defines storage or merely refers to storage defined. If you look at different C compilers, you will find that in most of them, "extern" means something different from what it means in the Bell Labs C compiler. (The meaning in the Bell Labs C compiler is presumably what K&R had in mind.)

Mr. Greco states that "The truth is that one dimensional arrays and single indirection pointers are the same internally." This is NOT true as arrays and pointers defined outside a function are quite different. It is strange that Mr. Greco would say this, since elsewhere in the article he notes that an array is a constant pointer, whereas other pointers are not necessarily constant.

Mr. Greco claims it is incorrect to say that

```
zippo == & zippo[0][0]
```

if zippo is a two-dimensional array defined by

```
int zippo[4][2];
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

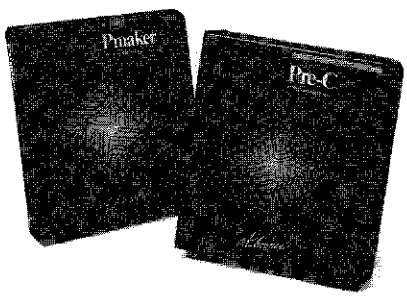
Finally, A Lint and Make for MS™ – DOS

Get the full range of features C programmers working in UNIX™ have come to expect from their Lint and Make utilities. With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. Pre-C outlints Lint, since you can handle analyses incrementally.

Pre-C's flexible library approach lets you maintain continuity across all programs in your shop, whether you use Pre-C's pre-built libraries, functions you already have, or some you might want to buy.

Plus, you're not limited to one particular library. Pre-C keeps track of all the libraries you're using to make sure that code correctly calls them.

With Pmaker™ you can update and track every module in your program. When you make a change in any source or include file, all you do is run Pmaker. It will recompile changed modules and relink your program. With any compiler or linker you choose. Pmaker can update an object module library when one or several of the object modules are changed. You can use Pmaker to handle any task when a change requires several steps.



Pre-C by Phoenix. $395. Pmaker by Phoenix. $195.

Call (1) 800-344-7200.
In Massachusetts (617) 762-5030.

Phoenix Computer Products Corp., 320 Norwood Park South, Norwood, MA 02062

PROGRAMMERS' PFANTASIES™ BY PHOENIX

Phoenix

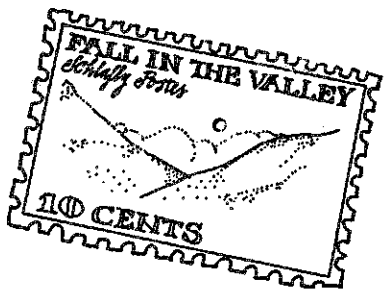
See us at COMDEX/Fall '85

Programmers' Pfantasies, Pre-C, and Pmaker are trademarks of Phoenix Computer Products Corporation.
MS is a trademark of Microsoft Corporation. UNIX is a trademark of AT&T Bell Laboratories.

above equality to be false is nutty; I would challenge him to find me a C compiler where this equality does not hold.

Roger Schlafly
Borland International
Scotts Valley, CA 95066

[Looks like someone at Borland is hard at work on a C compiler. —Editor]



Frank D. Greco replies:

My article claims that

```
int x;
x = ++x;
```

are questionable and potentially unreliable. I stated that "[this is] a dangerous construct that is not guaranteed to work the same on any two compilers." The key word here is *guaranteed*. Empirically, Mr. Schlafly may find that the C compilers that he knows produce the same effect (that is, incrementing a variable correctly); however, he is relying on evidence derived from his own experience, as opposed to the law of the language. If you do a lot of highway driving, the speed limit may appear to be about 65 MPH based on your observation that everyone drives at that speed (or higher on Highway 101 in the valley!), but the law specifically says that the speed limit is 55 MPH. The C language bible, K&R, says on page 50: "When side effects (assignment to actual variables) takes place is left to the discretion of the compiler, since the best order depends on machine architecture." Above this, K&R note,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"One unhappy situation is typified by the statement

```
a[i] = i++;
/* or 'a[i] = ++i;'   FG */
```

The question is whether the subscript is the old value of i or the new. The compiler can do this in different ways, and generate different answers depending on its interpretation."

I teach the following rule in my C language classes: When the same variable is used more than once in an expression and at least one of them is involved in a side effect, the result is not guaranteed to be portable.

Mr. Schlafly is certainly depending on order of evaluation here. The C language does not have a predefined order of evaluation (except for the left-to-right evaluation of the comma, &&, and || operators). K&R say (page 185) " . . . the order of evaluation of expressions is undefined." And on page 50 of their book, "The moral of this discussion is that writing code which depends on order of evaluation is a bad programming practice in any language." Effectively, since the programmer merely wants to increment a simple integer variable, he/she should merely say (as Mr. Schlafly points out):

```
int x;
++x;
```

Mr. Schlafly is right in pointing out the preferred construct; it certainly is helpful for many of my readers who are novice C programmers. Since the opening paragraph of his letter says I should not cater to "naive" readers, I wonder why Mr. Schlafly seems to here.

My intent in writing a direct and portable translation of Traister's programming segment was *not* to say that my version is the recommended method. Notice that I did preface my version with the phrase, "A possible corrected version would be . . . "

Regarding DEFINITION v.s. DECLARATION, I must admit that rigorously speaking, the difference is a question of semantics. The common usage of these terms has blurred their true meaning. Many C programmers say they want to "declare a variable" quite naturally because in most other languages (Pascal, FORTRAN, BASIC) declaration of a variable means to allocate storage for a variable. In addition, the normal (i.e., non-computing) usage of the words DECLARE and DEFINE dilute the formal meaning of the terms. However, knowledge of C's formalities will allow the student to thoroughly understand the language. Strictly speaking, there is no such animal, as Mr. Schlafly states, as "defining a data type" in the C language. That is known as a data DECLARATION. The following are examples of DECLARATIONS:

Instant-C™:
The *Fastest* Interpreter for C

Runs your programs 50 to 500 times faster than any other C language interpreter.

Any C interpreter can save you compile and link time when developing your programs. But only ***Instant-C*** saves your time by running your program at compiled-code speed.

Fastest Development. A program that runs in one second when compiled with an optimizing compiler runs in two or three seconds with ***Instant-C***. Other interpreters will run the same program in two minutes. Or even ten minutes. Don't trade slow compiling and linking for slow testing and debugging. *Only* ***Instant-C*** *will let you edit, test, and debug at the fastest possible speeds.*

Fastest Testing. ***Instant-C*** immediately executes any C expression, statement, or function call, and display the results. Learn C, or test your programs faster than ever before.

Fastest Debugging. ***Instant-C*** gives you the best source-level debugger for C. Single-step by source statement, or set any number of conditional breakpoints throughout your program. Errors always show the source statements involved. Errors always show the source statements involved. Once you find the problem, test the correction in seconds.

Fastest Programming. ***Instant-C*** can directly generate executable files, supports full K & R standard C, comes with complete library source, and works under PC-DOS, MS-DOS, or CP/M-86. ***Instant-C*** *gives you working, well-tested programs faster than any other programming tool.* Satisfaction guaranteed, or your money back in first 31 days. ***Instant-C*** is $495.

Rational
Systems, Inc.

P.O. Box 480
Natick, MA 01760
(617) 653-6194

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
/* No storage is allocated */
struct car
{     int weight;
      char color;
      float price;
};

/* No storage is allocated */
union flag
{     int count;
      float cost;
};

/* No storage is allocated */
enum color
{     red, orange, yellow, green,
      blue, indigo, violet
};

/* No storage is allocated */
typedef char * CHARPTR;
```

In reference to the last example, I would add that on page 200 of K&R, they say, "Declarations whose 'storage class' is typedef do not define storage, ..." Note that K&R say "Declarations," not "Definitions." The following are examples of data DEFINITIONS:

```
/* Storage is allocated */
struct car porsche;
/* Storage is allocated */
union flag book;
/* Storage is allocated */
enum color house_color;
/* Storage is allocated */
CHARPTR cptr, midptr;
```

I believe it is perfectly clear in K&R (page 76) that "A declaration announces the properties of a variable (its type, size, etc.); a definition also causes storage to be allocated."

Mr. Schlafly is incorrect in his statement that "K&R says that data declarations [!] outside function definitions are assumed to have the storage class specifier 'extern' if none is given." Page 204 of K&R (Section 10) states, "An external definition [note that K&R say 'DEFINITION'] declares an identifier to have storage class extern (by default) or perhaps static, and a specified type." Mr. Schlafly reasons that according to this rule,

```
int x;
```

and-

```
extern int x;
```

should have the same meaning outside a function. This is an erroneous assumption. Assuming they are both syntactically outside any function, the first one is a data DEFINITION (hence storage is allocated) and is of storage class extern; that is, this is an external data definition. In the second case, K&R state (page 193), "In the extern case there must be an external definition for the given identifiers somewhere outside the function in which they are declared."

To compiler writers, how to lexically differentiate the actual definition of external variables as opposed to their declarations is a familiar problem. In Harbison and Steele's *A C Reference Manual*, they state that the most common solution to this problem for compiler authors is to leave off the extern keyword on the DEFINING statement and to include the extern keyword on the DECLARING statements.

Mr. Schlafly errs when he says, "The meaning in the Bell Labs C compiler is presumably what K&R had in mind." Kernighan and Ritchie did *not* write about the Bell Labs C compiler in their classic. Their book describes the specifications of the C programming language on *any* machine. Besides, the AT&T Bell Laboratories C compiler is based on pcc, the portable (!) C compiler.

Mr. Schlafly points out that my statement on the sameness of one dimensional arrays and single indirection pointers is incorrect if they are defined outside a function. I was commenting on Mr. Waite's (*C Primer Plus*) statements that arrays and pointers are completely different. Well, I must admit my phrasing of the concept was a bit imprecise. Obviously an aggregate data type (e.g., an array of 100 chars) is not the same as a pointer (i.e., a single machine address) inside OR outside a function. I certainly hope that my readers did not think that I meant that! I should have said, "Arrays are converted internally to pointer expressions." In the C language "array subscripting is defined in terms of pointer arithmetic" (Harbison and Steele). K&R says on page 210, "By definition, the subscript operator [ ] is interpreted in such a way that E1[E2] is identical to *((E1) + (E2))."

Mr. Schlafly misquotes me when he says, "Mr. Greco notes . . . that an array is a constant pointer, whereas other pointers are not necessarily constant." In my article, I said (*PJ* Vol. 3 No. 4 page 17), " . . . the name of any array is a pointer constant. It points to the zeroth element." Certainly, an array is not a pointer. See my comments above in that regard. When it comes to pointers in the C language, it is a truth to say that there is nothing special about them. They are merely addresses. Just as there are int variables and int constants, char variables and char constants, and float variables and float constants, there are POINTER VARIABLES and POINTER CONSTANTS. A pointer variable is capable of storing a pointer constant.

Again Mr. Schlafly misquotes me when he says "Mr. Greco claims it is incorrect to say that

```
zippo == & zippo[0][0]
```

if zippo is a two dimensional array defined by

```
int zippo[4][2];
```

What I did say (page 20 of *PJ*) was that the *C Primer Plus* is incorrect in its assessment that the name of the two dimensional array 'zippo' always "points to the first column, first row" (page 372 of the Waite book). The truth is that 'zippo' points to the zeroth row, which is an aggregate. Again, according to K&R on page 210,

"For example, consider

```
int x[3][5];
```

Here x is a 3 x 5 array of integers. When x appears in an expression, it is converted to (the first of three) 5

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

membered arrays of integers."

K&R are saying that since the array above is a two dimensional array, it really is an array of arrays. Thus, x is a pointer to the zeroth array (of three). Nowhere do they say that it MUST point to the zeroth entry of that array.

Another axiom that I use when I teach: The name of an array by itself is a pointer to the zeroth element. In the case of multi-dimensional arrays, the zeroth element is another array. Mr. Schlafly is once again relying on observation instead of reading the bible (K&R) in determining the actual address that the array name evaluates to. What may be confusing Mr. Schlafly is the rule that if you apply the indirection operator to the name of a multi-dimensional array, then you are GUARANTEED of retrieving the contents of the zeroth row, zeroth column. As I state in my article, although K&R says that the array name is a pointer to an aggregate, it is up to the compiler writer to select which element of the zeroth row to *actually* point to. This was done so that compilers can take advantage of the underlying machine architecture. While it is terribly masochistic to do so and consequently not as likely to occur, a compiler writer can select the address of the middle element of the zeroth row as the actual pointer. Given most of today's popular cpu architectures, it makes sense for compiler authors to have the two dimensional array name point to the zeroth row, zeroth column, but "it ain't necessarily so."

I would add for those application programmers learning the C language, that the exact value of a pointer is irrelevant for the vast majority of applications.

I do appreciate that Mr. Schlafly took the time to write to *PJ* concerning my article. Anyone who is so inclined is invited to write to me.

Frank D. Greco—Redirections
P.O. Box 470
Kearny, NJ 07032

IS *PJ* A "C" MAGAZINE?

Richard Larratt, one of your contributing writers, introduced me to *Programmer's Journal* with a copy of his article on the future of disk protection. I immediately thought this was my kind of magazine, hence, the subscription. The only advertisement showing within his article was a half-page from Greenleaf on "C". This issue struck me as being very "C" oriented with about 3½ full pages of ads for "C" versus 7 for all other products. Your "Redirections" column reviews 24 "C" books this and next issue, and the articles discussing "C" seem to be everywhere. Are you sure the subtitle is correct? Try "The Resource Journal for IBM "C" Programmers."

Brian R. Davis
Halifax, Nova Scotia, Can.

You're right, we've written a lot about C. It's a powerful programming environment, but not the only one. Look for more on Pascal, Assembler, and BASIC in the coming year.

Modula-2 also deserves some attention.

I want to look at the fringe, too — Smalltalk, Prolog, Lisp, Forth. One industry wheel recently called AI "a bunch of crap," but we could all use a little fertilizer. —Editor □

Write it once!

MasterFORTH
Portable programming environment

Whether you program on the **Macintosh,** the **IBM PC,** an **Apple II** series, a **CP/M** system, or the **Commodore 64,** your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	125
CP/M 2.x (IBM 3740 8")	125
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$60
Graphics (Apple II series)	60
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 FORML Conference	20

MICROMOTION
12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C's HLL OVERHEAD

Excerpts from DTACK GROUNDED: The 68000/32081 Systems Journal

by FNE, © 1985 by Digital Acoustics, Inc.

Daniel Lunsford's writeup in the last issue of *DTACK GROUNDED*, in which he claimed that the abysmal performance of existing application programs written in C was due either to poor UNIX-based run-time I/O libraries or to poor algorithms received much favorable notice, mostly regarding his assertions about the run-time libraries. For instance, an anonymous source at Sun Microsystems tells us that Sun rewrote its I/O libraries and got a 30% to 40% speed improvement in its programs. However, there is a lot more than 30% or 40% overhead in WORDSTAR 2000, DRI's Personal BASIC, and other such application programs and languages written in C.

> The other day, Lee Felsenstein, who has a LOT of experience with 8080/Z-80 micros, told us about his first encounter with a 16-bit microcomputer enthusiast. Lee asked, what do we need 16-bit computers for? "To get a larger address space." Why do I need a larger address space? "So you can have a lot more RAM." Why do I need a lot more RAM? Lee pursued. "So you can run programs written in high level languages." Why do I need to run programs written in high level languages? "So you can hire cheaper programmers!" the 16-bit enthusiast concluded.

Daniel L. would have us believe that the ONLY other source of HLL inefficiency aside from the I/O libraries is bad algorithms, but we don't believe that! (Sorry, Dan.) For one thing, it leaves open the possibility that the ASSEMBLY LANGUAGE PROGRAMMER might be the one to choose the bad algorithms, which could leave us with a C-based application program which runs TEN TIMES FASTER than the equivalent application program written in assembly! In fact, there are no such examples in existence (we are talking about real commercial programs).

Therefore, we need to look further into the question of C's HLL overhead. We begin with a letter from down under which apparently addresses the 90-10 rule but continues with the question of why C runs slowly compared to assembly:

"About the mythical 90-10 rule: The rule is correct, but the conclusion is a myth.

"The problem is that the programmer must represent the concepts, data, and algorithm of the problem that he wishes to solve in terms of the concepts, data types, and commands of the language he is using.

"The 'efficient' code produced by optimizing compilers is an efficient representation of the concepts, data types, and commands of the language, not necessarily an efficient representation of the problem that [the] language is itself representing.

"The task of the programmer is to represent the data and translate the algorithm into terms the machine can handle. The rest is just detail. Naturally this translation will be most efficient if the programmer translates into things that may be directly represented in the architecture of the machine, rather than things that are directly represented by the HLL.

"Most C compilers have about a 'two to one overhead,' in the sense that if you look at a small block of C code, it executes about half as fast as the same thing hand assembled. But if you were writing your whole program, or a major conceptual section of your program, in assembler, you would not express your solution in a way such that a small chunk of your program was directly equivalent to a small chunk of C code. In other words, an assembly language program is not 'the same thing hand assembled,' so it runs a hell of a lot faster than 'the same thing hand assembled' — usually about 15 times faster.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"As a result, your assembly language program would typically run about 30 times faster, not two times faster. In some cases the speedup can be as much as 1000 times, rather than two times. This is the case when your program is doing a lot of non-numerical symbol manipulation stuff.

"Because the speedup arises from the way in which concepts are represented, not from humans being more efficient than compilers, writing 90% of your program in a HLL, and then hand optimizing 10% in assembly can at most give a two to one speedup, and is generally not worth doing unless your HLL is interpreted rather than compiled.

"I repeat: Humans are not much better at compiling than computers, so it is seldom worthwhile to rewrite the 10% in machine code unless you are using an interpreter such as PASCAL or BASIC. The assembly language speedup comes from the way in which concepts are represented, not from the efficiency with which the representation is compiled; hence the 90-10 rule is irrelevant.

"By the way, some IBM mainframe FORTRAN compilers have a ratio of better than 1 to 1 — in the sense that they produce object code more cunningly optimized than most humans could manage. But because FORTRAN is extremely poor for representing many classes of problem, notably compilers, the effective ratio on that class of problem is about 400 to 1 — FORTRAN compilers written in FORTRAN are usually horribly slow and amazingly huge.

"Furthermore, in most large companies, the job of optimizing the 10% is given to some low-grade bloke, who has not got permission to change the way in which concepts are represented and (who) usually does not have the knowledge — he is just being used as an optimizing compiler. As a result, the speedup obtained by applying the 90-10 rule is usually insignificant whereas the speedup obtained by writing the whole thing in assembly is usually overwhelming.

"A program written in assembly will usually run about five to twenty times faster than an equivalent program written in C on an optimizing compiler with a 1.25 to 1 overhead and assembly language optimization of the 10% that takes 90% of the time, although this depends very much on the program, the programmer, and the way that the organization works.

"In a large organization where a number of programmers are organized in a hierarchical fashion, assembly language will frequently fail to produce a significant speedup because the inefficiency in concept representation derives mainly from the human to human interface rather than the human to machine interface; hence the tendency for large companies to use HLLs even in time-critical utilities."

James D., Dundas, Australia

James, we thank you for pointing out a way in which HLL languages inherently produce code which runs a great deal slower than assembly, optimizing compilers, or no optimizing compilers. Uh, we hope you do not mind that we agree with your comments completely. —FNE

Daniel L. kicked off this somewhat serious discussion of why C-based application programs run slowly [See *Programmer's Journal*, Volume 3, Number 4 —Ed.]. Here is additional input from Dan:

"Let's look at some things about compilers in general, and C in particular. There are two main ideas that dominate the design of any compiler, these being the design of the language being compiled (is recursion permitted? Are procedures implemented, and are they permitted to be nested? What control structures are there? Are there reserved words? Etc.), and the design of the target machine (is the architecture accumulator-based? Is there a hardware stack? Can data objects be easily tranformed into machine objects? Etc.). Looking at the first set of questions, we can see that they are in some sense analogous to the second set, in that the answers define what might be called a "pseudo-machine" (Pascal, anyone?) or "virtual" machine, whose opcodes can be considered as the primitives of the language (by the way, Felgercarb, what are your opinions on the Pascal Microengine? I really would like to know). A C-machine, for example, would have a "pre-increment" memory instruction that would correspond to the + + operator.

"What most compiler designers choose to do is concentrate on the virtual machine, rather than the target machine. This allows them to separate the compilation process into two phases, broadly speaking: 1) syntax checking, lexical analysis, and parsing, and 2) generation of an intermediate code (generally in the form of parse trees and flow diagrams) which is then worked on in a machine independent manner to produce more compact intermediate code. This is then fed into a code generator for the target machine, and sometimes (not nearly often enough, to my way of thinking) this is subjected to further manipulations, which shrink the result in a machine dependent manner.

"It should be clear, from this, that the closer the structure of the virtual machine is to the target machine, the better the code generated will be. At least, "better" in the sense that the program's semantics will be preserved in the least space on the target. This is one reason that C will never be the language of choice on the Z-80, say; the contortions necessary to implement the full semantics of a C program on the Z-80 are extremely painful and SLOW. On the 68000, however, we have machine primitives that correspond quite closely with the C primitives, so it's no surprise that good 68000 code can be gotten from C.

"Notice how I keep harping on the FULL semantics of a program; this is really the crux of a lot of misconceptions. To

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

use one instance, suppose that you want to compute 400! (factorial). In C, we may code this recursively, since C specifies that all functions are recursive (potentially). This potential recursion is part of the semantics of the program, since the meaning of "function" in C includes recursion. A compiler that ignored this would not be a valid compiler of the C language. An assembler programmer attempting to duplicate the full semantics would also have to implement the function in such a way as to allow recursion, or (put it another way) reentrancy. In short, I am saying that the assembler programmer who computes using non-reentrant techniques (which might well yield faster and/or shorter code) is not solving the same problem the compiler is!

"There are several other problems with various forms of 'optimization' (a misnomer, actually) on micros, and on mainframes as well, and some of them are really thorny. First- and second-generation languages such as FORTRAN and COBOL can be handled on a more or less local basis, since neither permits recursion, local data areas, etc. A compiler for these languages is free to do a great many things. I once read (I forget the exact reference; if you wish, I will attempt to locate it) of a FORTRAN compiler, one of the early versions of IBM FORTRAN H (I think), that made some very astonishing manipulations of the input to get the output. There was a test subroutine that did nothing, but computed a lot, whose purpose was to exercise the compiler. I gather it was a VERY sizeable routine. The people were very surprised to find that the output object code was just a few bytes! BUG! they thought. Very careful analysis yielded the interesting dictum that the compiler was doing just what it had been designed to do. It seems the compiler had looked at the subroutine, invoked some global analysis algorithms, and discovered that the routine had no input, no output, and no side effects; since the routine was not doing ANYTHING, the compiler THREW IT AWAY! Needless to say, this is not what was intended, since the compiler was not preserving the semantics of the program, in fact wasn't preserving the program at all!

"What you would like, I suspect, is a compiler that could detect when recursion (for example) is not necessary and generate the more compact code that would result. If all routines called in the program actually reside in the same source file, this is indeed possible. If not, you have some problems. Suppose that function A calls routine B. B in turn calls library function L. Then no analysis of the source file containing A can assure that L is re-entrant or not, and hence the compilation of the program must proceed as if it is. The compiler can hardly read the programmer's mind.

" . . . Your point on the 3B2/200's floating point routines is well taken. I agree completely that the UNIX I/O garbage could hardly affect floating point. But remember that I had TWO possibilities for C inefficiency: bad libraries and bad algorithms . . . SIGH. Please note that I am just as disgusted at those timings as you are." Daniel L., Carmichael, CA

PETER NORTON JOINS THE DISCUSSION . . .

I just got the July/August issue of *Programmer's Journal*, and I'm prompted to comment on the excerpts from *Dtack Grounded* that discussed the issue of the apparent efficiency or inefficiency of C and related operating system issues.

I've often pondered the issue of programming language overhead as many programmers do. It's seemed to me that there is a dichotomy in the use — or hoped-for use — of programming languages that causes confusion in how people think about programming languages.

Roughly speaking, there are two aspects to any high level language: code generation and the delivery of services. Code generation, of course, is the translation of the guts of our programs into machine language; the delivery of services involves things like memory allocation, I/O (especially high-level I/O that includes error handling, etc.), and so forth.

In a general sort of way we can divide up programmers (and programming tasks) into those that focus on a compiler's efficiency in generating code and those that focus on the quality and convenience of the services. One is concerned with not wasting the computing resource of the machine; the other is concerned with not wasting the programming resource of the person (by relieving the programmer of tediuos I/O programming, for instance). In a crude way we can say that systems programmers fall into the first camp, application programmers into the second.

Both schools of thought are valid and sensible and focus on a useful emphasis in programming. If we're sensible, though, we can see the importance of both aspects of high level languages. While I am very fond of reading the excerpts from *Dtack Grounded* that have been printed in *PJ*, I'm often annoyed by the myopic and sometimes seemingly fanatical adherence to the efficiency viewpoint. But it often seems that compiler writers are dedicated to the other camp: I'm always grumbling at compilers that insist on burdening programs with the overhead of high level services, error checking, and more.

In my programming career, I've often used and been grateful for lots of hand-holding from my compilers that saved me long hours of work by providing very high level services. I don't share the apparent bias of the folks at *Dtack Grounded* that those high level services are nothing but gross inefficiency. But I've also been on the other side of the fence, wanting to avoid compiler overhead.

A year ago, when I undertook to completely re-write my Norton Utilities programs from scratch, I was faced with the very issues that were brought up in the discussion "How Fast/Slow is C?" I wanted to use a compiler — particularly a C compiler — to give me efficient code generation (and relieve me of the common horrors of debugging error-prone assembly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

language), but I didn't want any of the typical library overhead that normally comes with high level languages.

Fortunately, I was easily able to solve this problem, using the Lattice-C compiler. First, I did what an efficiency-minded programmer can do with nearly any language and compiler: I completely avoided using any of the library routines that came with the compiler; instead, I wrote my own service routines, custom-tailored to do exactly what I needed and no more, writing them in C or assembler as appropriate. Second, I did what can only be done with cooperative compilers like Lattice-C: I avoided library overhead being drug into the program by carefully modifying the prologue-epilogue routines that the compiler uses. (This was possible because Lattice-C provides the source code for prologue-epilogue; for compilers that don't, the task is harder, but usually can be done with some effort.)

In short I was able to use a high level language for exactly the programmer-saving efficiencies that I wanted (generation of code, without my having to labor away in assembler) and avoided the programmer-efficiencies that I didn't want (by writing my own set of service routines). For my purposes, I got most of the best of both worlds: code reasonably close to hand-written tightness without having to write all that assembly and no ugly library fat to haul around.

That achieved pretty much what the "How Fast/Slow is C?" discussion was after, a clear proof that it can be done without too much sweat. But the truth of the matter is that as a rule compilers stack the deck against that sort of programming, acting as though high-service high-overhead programming is the only thing compilers should be used for. I've often believed that compilers should be — in some way — dual-mode, so that they could be easily used either way — either including or avoiding the use of library service routines. A literally dual-mode, lean or fat, compiler is probably not a practical beast, but we can get close to having one. With a language as inherently lean as C (so that ordinary code generation doesn't indirectly call for heavy library support) and with a compiler as open as Lattice-C (open in the sense that prologue source code is included), we can have it both ways.

Peter Norton
Norton Utilities □

PRODUCTIVITY UTILITIES

From Opt - Tech Data Processing

NEW **On-Line Help** NEW

A comprehensive utility for adding help windows to your programs. It provides an efficient method for storing help texts in libraries and standard routines for interfacing your programs with these libraries.

Help windows are displayed in a fraction of a second. You have total control over the contents of the window, its position on the screen and the display colors used.

On-Line Help can be interfaced with interpreted Basic and all popular compilers.

On-Line Help includes a Help Library Manager, Library Interface Routines and source code examples. **$149.**

Opt-Tech Sort

High performance sort/merge utility can be used as a stand-alone program or called as a subroutine to many languages.

Supports unlimited filesizes, multiple input files and fixed or variable length records. Many special file types are supported including Btrieve and Dbase. Up to nine sort control fields (ascending or descending), all common data types supported. Output file can be full records, keys or pointers.

Written in assembly language for high performance. Example: 4,000 128 byte records sorted to give key and pointer in 30 seconds. **$99.**

Scroll & Recall

Allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations).

Allows you to easily recall and edit your previously entered DOS commands and data lines without re-typing.

Scroll & Recall is very easy to use. It's a resident utility that's always there when you need it! **$69.**

All programs are fully documented and are compatible with the IBM-PC, XT, AT and compatible machines using the DOS operating system.

Visa, M/C, Check, Money Order, COD, or PO accepted.
Quantity and Dealer discounts available.

To order or to receive additional information write or call:

Opt-Tech Data Processing
P.O. Box 678 — Zephyr Cove, NV 89448
(702) 588-3737

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOW TO RECOVER ALL YOUR COSTS

by Paul Barkley

There's an insidious disease eating away at consultants, sapping their strength, diverting their energies, and in general trying to put them in the poorhouse. No, I'm not talking about addiction to junk food, caffeine or nicotine; I'm talking about something even more serious: unbillable hours. The very words send a shiver down my spine. Unbillable hours are the curse of the consultant, and if unchecked, can lead to rack and ruin.

I often think of the work that consultants do as analogous to giving a musical performance. People don't pay Willie Nelson by the hour to get up on stage. They pay him a set fee, which is outrageously high for a couple hours work, but this fee represents all the time spent *before* the concert, practicing his licks and writing his music. In the case of country and western stars, of course, it also represents the years spent getting drunk, cheating on spouses, and driving trucks — all *bona fide* occupational prerequisites in that line of work.

Similarly, people pay consultants high fixed fees, or high hourly fees, because they expect the consultant to have "paid his dues" (or her dues as the case may be). Computer consultants are expected to be fluent in several high level computer languages that everyone's heard of, like "COBALT", BASIC, and maybe "Pasqual," as well as several secret and mysterious languages like binary and hexadecimal. The consultant should have memorized the conflicting ASCII tables for a half dozen different computers and be able to get a printer to stop printing everything on the same line.

All of this preparation is *expected* (and necessary), but not an hour of it is billable time.

Most of these accomplishments, however, can be mentally written off as general preparation, similar to attending college or a trade school — particularly hours devoted to the fundmentals. Experience in specialized hardware or software can be similarly treated as experience brought to any new job, as if you were applying for a regular (i.e., employee) position somewhere as a cornflake packing supervisor after working four years as a low level cornflake packer.

You must also expend unbillable time on an on-going basis. I'm sure you spend a considerable amount of time reading pertinent periodicals — you're reading *Programmer's Journal*, for example. If your memory is as bad as mine, you probably index some or all of the periodicals that you receive. If you read only one or two

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

periodicals, you might charge it off to leisure time activities — after all, you wouldn't be in the computer business if you weren't inherently interested in computers.

If you're really serious about keeping up, though, you probably read a number of periodicals and buy an occasional book, as well. The last time that I was foolish enough to keep track, almost 4000 pages of materials were coming across my desk every month in the form of subscription periodicals. Fortunately, many pages of the thicker magazines are repetitive ads that you need to read only once (maybe lightly), or can skip entirely, and many articles are far enough removed from your field of interest that they, too, can be skipped. That still leaves a lot of material to read, and it can't be lighly dismissed as recreational.

How many hours a week do you spend on reading? I average about 6 hours. (By the way, if you're not keeping a very accurate time log, you won't have *any* idea how much time you're spending — you're going to be in for a surprise.)

Professional preparedness goes way beyond reading, of course. I regularly charge off several hours a week to exploring programming concepts. For example, when dBASE III arrived, I spent time learning about the poorly documented memo field — time that paid off later when I could intelligently integrate this knowledge into someone's unusual application. Most consultants have an innate curiosity and willingly explore new packages and try out different techniques on old problems. Once a year or so, I do a partial rewrite on my original quick and dirty label program. I don't know anyone who hasn't written a label program — mine started off years ago as spaghetti code when I was in a hurry for a few labels and gradually evolved as I learned the meaning of the word "structure." It's a harmless place to try out new coding ideas.

Other consultants spend a few days a year in professional seminars to hone their skills and acquire new ones. Clearly, there are a number of ongoing tasks that are a necessary part of being a consultant, but which can't be billed. These comprise the second area of unbillable time (the first was your original preparation).

The third area is your overhead — the hours you spend in office procedures. It takes time to process the daily mail, fill out all those government forms, pay your bills, and talk to migratory insurance salesmen who get your number. It's easy to spend four to six hours a week in these activities. Some consultants bill their clients for the time spent preparing their bill — but that's asking for trouble in my opinion.

The fourth area is marketing. Contract programmers often don't do any marketing until they're "on the beach," but almost everyone else spends several hours a week making exploratory phone calls, writing proposals, and trying to close sales. These often unpleasant tasks are what keep the business moving and can't be overlooked (unless you're so great that you routinely turn away business — if you are, raise your rates.) If you're like many consultants, you'll have to regularly schedule marketing time — otherwise, you won't get around to it. I hate to do marketing because a lot of the activities come down to plain old selling. If I wanted to be a salesman, goes the consultant's reasoning, why did I go into computer consulting instead of hawking encyclopedias?

There's almost nothing you can do about the first four areas of unbillable time — that's why your *billable* hours are charged at a relatively high rate. If you try to charge your clients directly for these hours, you're going to get in trouble (some people doing government work this way get sent to jail). If you try to skip these activities, you'll find yourself falling behind in technical skills and information and losing business as other people out hustle you.

The fifth area, though, *is* one that you can do something about. This is the time spent with clients that sort of falls through the cracks and can't be billed. Not everyone has this problem, but let's look at typical scenarios that create these conditions.

When I first started consulting, I was thrilled to have any clients at all. In order to get a new client, I would often spend several hours in preliminary meetings, not only selling myself but also doing the initial analysis. Four hours could go by pretty quickly.

The situation would rapidly deteriorate. The client would want a fixed fee price because he or she couldn't afford to write me a blank check. In addition, I didn't have enough experience to accurately estimate the length of a project, AND there was no set of specifications at this point. So, in order to quote a fixed fee, I had to perform enough of a systems analysis to know what I was supposed to deliver. Of course the client expected me to do the systems analysis as part of the whole package — so far they hadn't paid me a dime.

Now, I had the choice of doing a proper analysis, or doing a cut-rate job. If I took the time to do the analysis properly, both the client and I would know exactly what I was supposed to deliver, and ultimately, there were fewer misunderstandings. If I cut corners, there were always surprises. Unfortunately, I was a lousy estimator. Even with an excellent set of specifications I often had outrageous two and three hundred percent time overruns — all of which I had to eat. The alternative was to try to renegotiate, or to walk off. Renegotiation, in my opinion, implies that you screwed up and want the client to pay for it. Walking off means you won't see the balance due you of course, and you'll quickly get a very bad reputation. So, I ate a lot of jobs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Many consultants that I talk to have done one fixed fee job in their career, and have vowed never to touch another — they probably learn a little more quickly than I do. Interestingly enough, many consultants' estimating skills are inversely proportional to their technical skills. Top notch people have the worst record of estimating, probably because every job looks to them "like a piece of cake."

Simply jacking up the fixed fee to cover the time overruns as well as the initial analysis didn't work for me because my clients were extremely price sensitive. Most users don't understand why software should cost as much as hardware.

The solution that I hit upon came about only after several conditions were met:

1. By keeping very detailed records of how long it took me to do various phases of every project that I worked on, I finally learned to estimate properly.

2. I gained enough experience to shorten the amount of time necessary to perform both analyses and software development.

3. I developed a number of modules (we'll talk about these in another column) that could be reused, which increased the quality of my products and decreased the amount of time needed to complete the task — essential for fixed fee jobs.

4. I finally got enough business that I wasn't afraid to quote a fair price and let the client go elsewhere if they didn't like it.

All of these conditions were necessary. Even if you know what you're doing, you need enough work backlogged to be able to afford to let unprofitable business go.

My solution, which of course is not appropriate for everyone, is to spend an hour or so with a new client. I prefer to do this on the phone because most people won't spend more than an hour on a phone call while they're perfectly happy to tie up an entire afternoon talking. I charge this time off to marketing.

Next, I undertake the feasibility study and systems analysis. I give the client an estimate in advance, (often 20-40 hours for the type of projects that I deal with), and this is always at an hourly rate — in fact at my highest hourly rate, which is half again higher than my regular programming rate. The analysis is obviously crucial to any development that may follow and requires your special analytical skills. I get 50% of the estimated fee when I show up for the first meeting and collect the balance the day that I deliver the study. Often the client will want to sit and discuss the findings instead of reading them (makes you wonder sometimes why you bother), so I either bill out the additional time or put some time into the bill beforehand to cover this. (Any extra time put into the bill and not used should be refunded, of course, but no one has ever taken *less* of my time than I anticipated.)

When the client is satisfied with the system design (what goes into this is also the subject of a future column), I quote a fixed fee for software development, installation, and training. I usually throw in support, but many people charge extra for this although most clients can't see why they would ever need to call you (unless something goes wrong, which is no doubt your fault, they reason). Software maintenance is a hard sell for most clients.

A FULL C COMPILER FOR $49.95

The Eco-C88 C compiler is setting a new standard for price and performance. Compare Eco-C88's performance to compilers costing up to 10 times as much:

	Seive	Fib	Defref	Matrix
Execute	12.1 sec.	43.1 sec.	13.7 sec.	21.3 sec.
Code Size	7782	7754	7772	9120
Compile-link	76 Sec.	77 Sec.	77 Sec.	92 Sec.

Eco-C88 Rel. 2.20, on IBM PC with 2 floppy disks, 256K. Benchmarks from Feb., 1985 **Computer Language.**

Eco-C88 includes:
* All operators and data types (except bit fields)
* Error messages in English with page numbers that reference the **C Programming Guide** - a real plus if you're just getting started in C.
* Over 170 library functions, including color and transcendentals
* New Library functions for treating memory as a file
* User-selectable ASM or OBJ output (no assembler required)
* 8087 support with 8087 sensed at *runtime*
* cc and "mini-make" for easy compiles (with source)
* Fast, efficient code for all IBM-PC, XT, AT and compatibles using MSDOS 2.1 or later.
* Complete user's manual

If ordered with the compiler, the C library source code (excluding transcendentals) is $10.00 and the ISAM file handler (as published in the **C Programmer's Library,** Que Corp) in OBJ format is an additional $15.00. Please add $4.00 for shipping and handling. To order, call or write:

MasterCard VISA AMERICAN EXPRESS

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476 • 8:30-4:30

ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I receive one-third of the fixed fee when the contract is signed, one-third at some milestone that's usually about half way through, one-sixth when I walk in the door to install it, and one-sixth after testing and acceptance. Thus, the fees keep pace with the work performed, and I'm never in the hole for a lot of work performed but not paid for. (On the other hand, on several occasions I have been paid the entire sum up front, and this was bad because it made it hard to collect for changes that the client wanted later — they reasoned that they were so generous with their payment, surely I could bend a little too?)

This system works for me, and it covers MOST of the hours put in on a project. I still find a number of hours taken up during the course of a project with miscellaneous questions — usually procedures that we've already gone over, but which an anxious client has brought up again, and questions which come up after installation. Generally, though, this system succeeds in billing the client for the actual work performed; in making the client responsible initially for phone calls and other disruptive and time-consuming behavior, and setting a responsible tone for the follow-up fixed fee work; and in limiting the total project cost for the client's benefit. It's fair. It works for me, and generally limits the curse — of unbillable hours.

Now, let's hear how *you* limit *your* unbillable time.

Drop a line with your thoughts to:

Paul A. Barkley
The Barbican Company
6845 Elm Street, Suite 710
McLean, Virginia 22101-3896
CompuServe: 74776, 2526 □

Brand New From Peter Norton

A PROGRAMMER'S EDITOR

that's *lightning fast* with the *hot* features programmers need

A WONDERFUL EDITOR FOR PROGRAMMERS FROM PETER NORTON

THE NORTON EDITOR

only $50

Easily customized and saved ▲ *Split-screen editing* ▲ *A wonderful condensed/outline display* ▲ *Structured programming features:* ● *Auto-indenting for* Pascal *and* C ● *A* hot *"find matching punctuation" feature* ▲ *Great word features too:* ● *Paragraph reformat* ● *Word wrap, word action* ● *You can write your programs* and *your manuals with this one editor* ▲ *Mouse support, too* ▲ *Great for* assembler, *fabulous for* Pascal *and* C

A STEAL AT $50!

Direct from the man who gave you *The Norton Utilities, Inside the IBM PC,* and the *Peter Norton Programmer's Guide.*

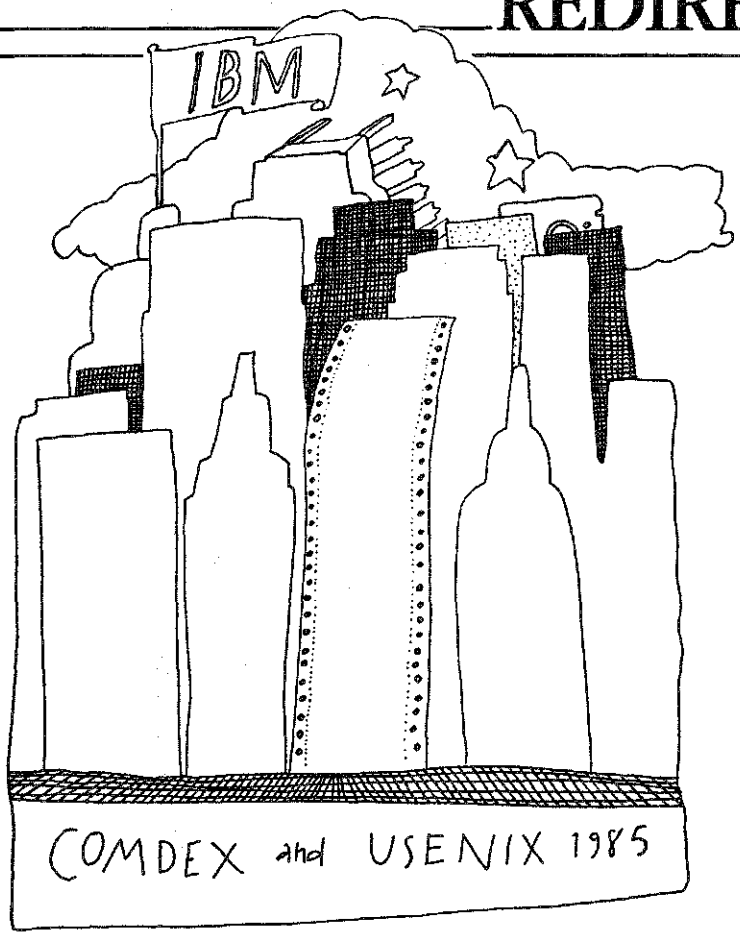
"This is the programmer's editor that I wished I'd had when I wrote my *Norton Utilities.* Now I've got it for you to use. You can *program your way to glory* with *The Norton Editor.*"

Peter Norton

Peter Norton, 2210 Wilshire Blvd., #186
Santa Monica, CA 90403, 213-826-8032
Visa, MasterCard and phone orders welcome

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

REDIRECTIONS



REPORT FROM THE COMDEX AND USENIX SHOWS

by Frank D. Greco

COMDEX

I enjoy the Comdex trade shows. Firstly, they are one of the remaining successful trade shows in the computer industry. For about 90% of the COMDEX week, it's nothing but business, business, business (the other 10%? . . . Stay tuned). As you walk up and down the massive exhibit halls, you hear things like "When can you have it running on my 68020 box?" "What are your royalty arrangements?" "What are your plans for the 386?" and "Where can you get aspirin and some Dr. Scholl's foot pads?" So as far as seeking out the proper VARs or OEMs for your product goes, "Youse came to da right place." In addition, where else can you see tomorrow's products today, and bump into people like Borland's Philippe Kahn and dBase guru Adam Green? This year's Spring Comdex in Atlanta, Georgia, was well attended with over 60,000 people. There were quite a few interesting exhibits (and exhibit booths). I'll try to give you some of the highlights.

"Please Nod Once for Yes and Nod Twice for No"

68000 UNIX box manufacturer Stride Micro announced a new input device called "The Nod." The Nod is a hands-free screen cursor controller. You attach a small device to your head (i.e., glasses) and position the cursor by moving your head . . . to boldly go where no mouse has gone before . . .

Stride also showed an impressive graphics board for their UNIX machines. The board is memory-mapped and retails for only $400. This super fast (I mean it!) graphics video board uses Stride's own standard terminals and yields a resolution of 784 by 325. Having that resolution on their standard terminals struck me as a little odd. I knew that Stride OEMs their fine terminals from publicly owned Wyse Technology. From past research, I knew that Wyse did not make graphics terminals. I queried the Stride technical representative and found out that Wyse had originally intended to market their terminals as graphics oriented. For some reason unknown to both me and the Stride tech rep, they altered one little jumper inside the terminal and made it into a character-only terminal. Needless to say, Stride determined which jumper it was and changed it. Now if only Wyse would do that . . .

Clone Zone

Televideo announced their new AT-compatible called "The Performance AT." Televideo's version of the IBM PC/AT uses an 8 MHz 80286 instead of IBM's 6 MHz 80286.

Compaq Computer Corp. showed their Deskpro 286 and Portable 286 (that's read "Transportable") at the show. Like the Televideo AT-compatible, these machines use the 8 MHz 80286 chip.

Speaking of 80286 chips, Intel announced 10 MHz and 12.5 MHz versions of the fast-becoming-if-not-already-popular piece of silicon. Now if only IBM used the 12.5 MHz version of the 286, PC/AT users would be getting over a 100% increase in speed! . . . sigh . . .

Now the VAR-sighted Altos, the dark horse in the UNIX race, waited for the faster Intel chips before announcing their impressive 286-based XENIX machine called the 2086. If their 986 and 586 XENIX units piqued your interest (*Byte* magazine called these two machines the best UNIX systems dollar-for-dollar), investigate the 2086.

Altos also has another impressive machine called the 3068 which uses the Motorola full 32-bit 68020 chip. In quantity, resellers can get a FAST (1.7 MIPS) multi-user 32-bit system starting around $10,000! You VARs out there should look into these UNIX-based systems. In the UNIX sweepstakes, Altos

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

came in second for shipped UNIX systems in 1984, following the market leader Tandy. They are projected to overtake Tandy in 1985.

Made In Japan

Hitachi had some impressive products at Comdex, including some high-quality color monitors ranging in price from $320 to $580. Their Compact Disk-ROM (CD-ROM) units were also there. Hitachi's read-only CD units hold 552 Megabytes per disk. These units cost $1930. A Hitachi representative told me that their user writeable CD units are about 1½ to 2 years away. Hitachi also showed their LCX PC-compatible lap computer. The LCD screen beats the pants off the Data General One; I was easily reading the screen from about 10 feet away AND I was at an angle from the unit. The same Hitachi rep told me to imagine this LCX lapper with a CMOS 68000 . . . . Then he was silent.

Sony also exhibited CD optical ROMs along with their new line of Trinitron RGB monitors for PCs. Be careful if you are considering buying the recently advertised RGB Trinitron monitors in some computer stores and retail outlets. These low-priced monitors are only RGB INPUT not RGB output! So while they accept a RGB signal from your computer, the display is still the lower resolution composite output. However, at the show, the true RGB Trinitron monitors were very impressive; the colors were bright and clear. The color black is JET BLACK on these screens (unlike the Princeton Graphic units).

Look Sharp

The Sharp Corporation showed an impressive array of computer products. Sharp has always been a competent technical company who appear to be still trying to pass Marketing 101 (American style, that is). Besides their amazingly powerful but cheap pocket computers, they demonstrated a powerful color scanner/digitizer. A very polite Japanese salesman (he hardly spoke English) proudly exhibited the GPIB (General Purpose Interface Bus) machine and told me to expect a super low price. I couldn't believe the speed and color accuracy of the digitization. He also pointed to another Sharp demonstrator showing off a tiny serial/parallel plotter. This 4-color plotter is a mere 13" by 6.3" by 3" and will retail for under $200! For that price, the resolution was tremendous. Now you can create presentation-quality color graphics using a cheap little plotter at home. "From Sharp Minds Come Sharp Products that need Sharp Marketers."

A while back I raved about Toshiba's letter quality matrix printers in my Redirections column. I reported that if you wanted speedy draft quality, graphics and letter quality from one printer, you should investigate the Toshiba P1350. Well, Toshiba showed their newest version of their slightly pricey printer called the P351. It makes all the Epson's fonts look like draft quality. With downloadable fonts and font cartridges, the P351 gives you 288 cps in draft mode and 100 cps in letter quality mode. Three fonts are ROM resident including proportionally spaced prestige elite. With at least three third-party vendors (plus Toshiba) creating libraries of fonts, the user has plenty of fonts from which to pick and choose. Although Toshiba lists the P351 at $1895, you can find mail order houses selling this superb printer at around $1000. Expect even lower prices around Christmas when overstocked inventories have to be moved.

Is It Soup Yet?

Microsoft showed off their surprisingly quick (in execution, not in delivery!) Windows package, which was demonstrated with the graphics help of a Tandy 2000. In addition they also introduced Excel, a spreadsheet for the 512K Macintosh. Excel has 16,384 rows and 256 columns available to the user and has a dynamic macro recording facility. Since Jazz is turning out to be a slow Waltz, Gates and crew are jumping on the opportunity to be THE spreadsheet on the Mac. I have a feeling that they will succeed. If you recall, when Bill Gates hired Jon Shirley away from Tandy, I wrote in this column that Microsoft will be more application oriented instead of system oriented. Mr. Shirley has an extensive retail background, and I'm sure *Wunderkind* Gates sees the potentially lucrative Window(s) of opportunity.

One personal observation about Windows: Microsoft's Windows uses the tiling (non-overlapping) approach to windowing as opposed to the overlapping, put-anywhere windows popularized by the many UNIX workstation vendors. It seems to me that the tiling approach is less flexible but faster (without any graphics hardware assist). The prevailing argument for tiling is that it's a hell of a lot easier to implement. Anyone who has designed his or her own window manager knows that the data structures are much simpler and faster than the overlapping windows'. The Microsoft PR crew did a good job in publicizing the tiling approach as a "new" windowing method; whereas in fact, tiled windows are what they are because they are EASY to implement. Since much of the delay in the Windows product has reportedly been due to the effort to make it considerably compatible with the Apple Macintosh, I wonder how Windows' tiling approach fits with the Mac overlapping windows philosophy? Besides, overlapping windows do not necessarily have to overlap.

The South's Gonna Do It Agin

Hometown kings Quadram announced Quadgraph, a board that performs graphics on your IBM monochrome screen. "With the proper software" (I love that phrase), you can get up to 720 by 348 resolution. I tried to find out whether this board needs drivers like the Hercules board or if it is similar to the Paradise and STB boards, which do not need any software drivers. No one at the surprisingly large Quadram booth

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C

HOW EASY DEVELOPING PROGRAMS CAN BE WITH THE C UTILITY LIBRARY!

COMPLETE. Over 300 tested and well documented functions. All features of the PC &AT are at your finger tips.

ADVANCED. Features like windows, data entry fields, switching displays, and batch file execution are supported.

GRAPHICS. A complete set of fast business graphics functions. Our low level graphics routines are the fastest in the business.

SOURCE. All source code is included. 95% of the library is written in C. Only functions demanding fast execution speed are in Assembler.

COMPATIBLE. With Lattice, Microsoft(3.0), CI-C86, Aztec, DeSmet, Wizard, and Mark Williams. All memory models are supported.

MUCH MORE. Dozens of string functions, the best time & date math and formatting, DOS directory and file mgmt., keyboard control, polled async communications, and more. **NO ROYALTIES.** We **SUPPORT** what we sell.

C UTILITY LIBRARY - $185
COMPILERS: Lattice $349, CI-C86 $329, Mark Williams $449. **Save $40-$50** with compiler & library package.

Specify compiler and version number when ordering. Add $4 for UPS or $7 for UPS 2-day shipping. NJ residents add 6% sales tax. VISA, MC, Chks & qualified PO's.

ESI Essential Software, Inc.
P.O. Box 1003
Maplewood, New Jersey 07040
914/762-6605

CIRCLE NO. 140 ON READER SERVICE CARD

could answer my question. I have yet to see the board advertised in any computer related publication since Comdex. This may be one of the many announced products that never see the light of the commercial day.

OS-A-MATIC: It Slices, It Dices . . .

Another Comdex exhibitor, Multi Solutions Inc. (609-896-4100), the people who are calling MS-DOS a toy and UNIX a dinosaur in their ads, have already formed a subsidiary to port applications software to their self-touted operating system called "S1." John Little, the director of systems software development for this tiny company in Lawrenceville, New Jersey, has often written articles on what seems to be a perfect operating system. It is multi-user, multi-tasking, true multi-processing (up to 256 processors), has a table-driven (hence alterable) process scheduler, has system calls for bit-mapped graphics, windowing, sophisticated inter-process communication (events, locks, messages, conditions, semaphores and gates), and networking, is source compatible with UNIX, can read AND write files to practically any other OS, is portable to ALL machines (they say from Z-80 micros to IBM mainframes), and is extremely modular. Well, what seemed to be mere chest pounding is actually turning out to be something substantial. Fujitsu has forwarded more than a few million to Multi Solutions to port S1 on their minis and mainframes. Our Department of Defense is also very interested in S1, according to the boastful company. Only time will tell . . .

3M of Minnesota (1-800-328-1300 ext: 3-2142) showed a videodisk information retrieval unit capable of 800 Megabytes in 5¼ inch form that will cost $800 in OEM quantities at the end of this year. They are also offering a videotape to videodisk service for a very nominal fee. The setup charge is about $5,000 and pressings about $10 to $18 dollars per disk. So for about $7,000 your company can have your internal training films digitized and ready for computer control.

There was a product called dFlow from WallSoft Systems, Inc., of New York City, that searches for and finds bugs in dBase progams instantaneously. I have a better idea; it's called dUser. It finds bugs much faster . . .

Stick It To 'Em

Video7 (408-943-0101) of Milpitas, California, showed their Mouse Stick, a joystick for the Macintosh. With the Mouse Stick, you can use the Mac mouse or the Joystick interchangeably. Apparently, Video7 built a device with two inputs that plugs into the mouse port of the Mac. You plug the mouse and the Joystick into this device. It allows you to play arcade style games on your Mac.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Write 'em Cowboy!

Addison-Wesley announced a micro version of the highly regarded TeX (Tau Epsilon Chi) technical text processing system designed by Stanford Professor Donald E. Knuth. Reportedly, Knuth was dissatisfied with the quality of computer typography that was available for his classic Art of Computer Programming series of books and created the easy-to-use TeX. Although its forte is mathematical typesetting, TeX can be used for general documentation. MicroTeX is a full implementation developed by David Fuchs for the IBM PC and compatibles. It retails for $495 and requires 512K of RAM and a hefty 1.5 Megabytes of hard disk space. The source code is in the public domain and is available for a nominal fee from DP Services, 1371 Sydney Drive, Sunnyvale, CA 94087. TeX is written in a language called WEB that "compiles" to Pascal. Unfortunately, the current version of TeX is batch-oriented and thus non-interactive. I was told that more modern interactive versions are now being written. If you want more information, contact the TeX User's Group (TUGboat) at PO Box 6248, Providence, RI 02940, or you can purchase Knuth's *The TeXbook* at $14.95.

Now About That Remaining 10% . . .

Any Comdex is not ALL work and no play; this one was no exception. Quadram sponsored a Crystal Gayle concert for invited guests, dealers, and media reps. They also had country humorist Jerry Clower. In addition, Disco singer/musician Teena Marie was in town for the occasion.

Atlanta's Limelight disco (yes, it's the same owners as the ones in NYC and Chicago) seemed to be the favorite place of most of the attendees. This histrionic dance palace was packed with mostly Comdex exhibitors and attendees. How did I know? Well you know those computer people, they all inadvertently left their Comdex identification badges on! In one corner I noticed two IBM reps trying to pick up a couple of AT&T people I recognized; it just seemed incongruous.

While the surprisingly inexpensive Limelight (I'm used to the Northeast economy) was a hit with many of the show goers, the place to be was Confetti's. It was considerably smaller than the monstrous Limelight, but had a more intimate ambience. With the flowing liquor and dancing feet, I just wonder how those exhibitors felt the next morning. All you Ashton-Tate people know what I'm talking about. And to think that all you thought they did was work on dBase!

AND NOW FOR SOMETHING THAT'S COMPLETELY DIFFERENT . . . USENIX

The Usenix conference/trade show is geared towards the UNIX oriented academic community. This Summer's conference in Portland, Oregon, was one of the best ever; it marked the tenth anniversary of the group. While Comdex-goers talk of "seeing black this quarter," "consummating the VAR deals," and "cross-licensing arrangements," the Usenix attendee hears "point-to-point TCP/IP Network links," "virtually linked filesystems," and "window based symbolic debuggers." The conference proceedings of this technical rap session was the largest one ever assembled; it was over 600 pages.

Many big names in the UNIX community attended. Dennis Ritchie (inventor of the C language), Rob Pike (authored software for AT&T's BLIT bit-mapped graphics terminal), Bill Joy (senior architect of Berkeley UNIX and author of the "vi" screen editor), Alan Feuer (author of the *C Puzzle Book* and the Safe-C interpreter), Dave Korn (author of the newly released Korn shell), and other techies traded ideas, algorithms, and implementations. There was a ton of useful information for UNIX hackers at the show; I'll summarize the highlights.

Getting Faced

Two AT&T Bell Labs researchers, Rob Pike and Dave Presotto, discussed their "network face server." Digitized face images of selected Bell Labs MTSs (Members of Technical Staff) and others were stored on one Eighth Edition UNIX machine in Murray Hill, New Jersey. What is so interesting about this

INTRODUCING
DATALIGHT C

The Datalight C Compiler for MSDOS is a full C with all K&R constructs, including bitfields, plus the version 7 extensions. Other features of the compiler are:

$60

- Produces object files (.obj) so just the MSDOS linker is required.
- Floating point performed with 8087 or automatic software floating.
- Over 100 compact library functions with source.
- Compatible with the Lattice C compiler.
- Runs on IBM-PC, and compatibles, running MSDOS 2.0 or later.
- Complete, easy-to-read users' manual with index.
- Highly optimized code for production quality programs.

Datalight
11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

Outside USA add $10 shipping. Washington State residents add 7.9% sales tax. VISA and MasterCard accepted.

IBM-PC, a trademark of IBM; MSDOS, a trademark of Microsoft Corp.; Lattice C, a trademark of Lattice Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

server is that it appears as a variant of the network filesystem. In this way, the images are centralized on one machine but appear to be regular files in the filesystems of ALL the machines on the network. While this light-hearted application seems ludicrous, the researchers say that the theory can be applied to any database, textual or graphic.

Hmmmmm . . . Ice C Now!

Another Bell Labs MTS, Andrew Koenig, described the use of a new language called "SNOCONE." It provides all the power of SNOBOL4, like dynamic data typing and superb string handling facilities but with the added "syntactical sugar" of the C language.

Dennis Ritchie's New Brainchild — Streams

Dennis Ritchie discussed his clean method of divorcing terminal processing from the terminal device driver. His method embodies something that he calls "streams." While Ritchie has been mentioning Streams for the past few years, he is now actively promoting them. They are currently implemented on the Research version of UNIX (Eighth Edition UNIX). You can think of streams as bi-directional pipes (a pipe is a method of connecting the standard output of one command to the standard input of another — command redirection).

Streams are seen by the programmer as file descriptors (similar in philosophy to MS-DOS 2.0's file handles) where the normal system calls still apply. In this manner, network protocols, terminal processing, and device drivers are separated cleanly. Now that they are separate, the network protocol of your choice can be "slipped" between a process and the terminal device driver. This would greatly improve the networking capabilities that UNIX already has. In addition, Ritchie has already stated elsewhere (Bell Labs Technical Journal, October 1984) that not just network protocolscan be inserted dynamically, but any program can be connected directly to other programs, providing interprocess communication. This would facilitate windowing and inter-machine processing.

At a panel discussion involving streams, which included AT&T Bell Labs colleagues Rob Pike and Dennis Ritchie, Rob asked the inventor of streams jokingly, "If they are bi-directional, why do you call them streams?" To that, Dennis Ritchie flipped the bird in mirthful defiance to his cynical co-researcher.

If the Price is no Object

Silicon Graphics (415-960-1980) of the ever popular Mountain View area of California, has one of the most truly exciting state of the art graphics workstations. Their system, which "starts" at $30-40K, uses their so-called Geometry Engine hardware for real-time graphics manipulation. One speaker showed a videotape demonstrating the powerful features of the system. The tape showed a screen editor scrolling in a window. The smooth scrolling was so fast, it seemed faster than page flipping on a PC/AT. Another part of the tape showed two windows on the workstation screen; one window displayed a realistic looking Macintosh-like mouse that moved a graphics cursor in another window, which drew and painted by itself. The movements of the mouse were extremely "lifelike." At their booth, the two ongoing demos included their spectacular color flight simulator and the 3 dimensional molecular model (you can "fly" through the molecule). If you would like to see a demonstration of this phenomenal machine, call 1-800-556-6661 and ask for the videotape (VHS or Beta). It is the fastest graphics system I have ever seen.

Here Comes the Sun

Sun Microsystems, as usual these days, were exhibiting their new workstations that use the Motorola 68010 microprocessor and the Berkeley 4.2 version of UNIX. Rumors flew around the booth that Sun is waiting to complete a contract with the government before going public. With graphics workstations being touted as the next high growth area, Sun is strategically positioning themselves as one of the leaders. I was told that the next revision of the high resolution monochrome model will be priced around $5,000. I recall reading somewhere that Carnegie-Mellon University is requiring all incoming freshmen majoring in any engineering discipline to buy one. Of course they would receive a reasonable discount from Sun.

Usenix is not a show for end users. It is a technical forum that brings together some of the top UNIX hackers in the world. It certainly was diametrically opposed to the Comdex show. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOMETIMES GOOD IS JUST NOT GOOD ENOUGH.

Frankly, there's a lot of good development software on the market today. But when your programming project is on the line, good isn't good enough. You need the best.

Because we're an independent publisher and distributor, we can pick and choose. We've been selling to microcomputer programmers since 1976, so we know quality when we see it. Lifeboat offers each of the following popular software development tools and more... When we offer a product, you know it is the very best in its class.

MARK WILLIAMS C PROGRAMMING SYSTEM™ New!

When programming, it takes an incredible amount of time just to get the code written. Then comes debugging, which can take even longer. *Mark Williams C Programming System* combines a C compiler with an easy-to-use debugger to make C programming simpler.

One-step compilation produces an executable file with a single command.

Options include object file in MC- or MS-DOS formats or ROM-able code.

Full UNIX capabilities, with large library of more than 100 routines.

Debugging facilities allow setting tracepoints at source level; windowing enables source to be viewed while debugging; optional assembly listing.

MWC compilers are available for 8088/8086, 80186, 80286, 68000, Z8000, and PDP-II. Vax cross compilers are available.

RUN/C LITE™ New!

Learn C the natural way with the *RUN/C Lite* introductory package. Over 100 sample programs are included on the disk to coincide with the thorough and professionally written C reference manual. *RUN/C Lite*'s user interface is similar to that of the BASIC interpreter with easy familiar commands such as LOAD, LIST, SAVE, EDIT, and RUN. Debug with TRON, TRACE, and PRON. Try your hand at graphics with easy-to-learn functions like PUT, GET, CIRCLE, LINE, SCREEN, and DRAW. Lighten up on the time it takes to learn C with *RUN/C Lite*.

RUN/C PROFESSIONAL™ New!

All of RUN/C Lite's features, plus powerful enhancements for program development. LOAD and RUN, your favorite library functions (GSS Drivers®, Multi-Halo™, Panel™, Lattice Window,™ etc.) from within your RUN/C™ interpreted code. Polish program modules before compiling. New source code debugging facilities allow you to set multiple breakpoints, single step through a program, and view and change variables during execution.

BetterBASIC™ New!

BetterBASIC is just what it sounds like. Better than the BASIC you got with your PC. What's better about it? Just about everything.

Fast. Programs run fast. Each line of code is compiled as it is entered into memory.
Big. Break the 64K barrier. Use the full memory of your PC to write big programs.
Powerful. Windows and graphics, DOS and BIOS calls, chaining, overlays, local and global variables, recursions and more.
Interactive. Responds to user commands in immediate mode.
Structured. Write structured programs using functions and procedures.
Modular. Create re-usable "library" modules for more efficient programming. Easy interface to assembly language.

BetterBASIC Runtime System,™ 8087/80287 Math Module,™ and Btrieve Interface™ also available.

Lattice® C Compiler

The best selling C compiler!

- *Lattice C* produces fast and compact code.
- The widest selection of C support tools is available to the *Lattice C* user.
- Choose from four memory models for optimum efficiency.
- Full compatibility with the C implementation established by Kernighan and Ritchie assures portability.
- Superbly written documentation augments the lengthy list of reasons why software developers continue to choose *Lattice C*, the industry standard.

GSS® PRODUCTS New!

Industry leading graphics software based on a commitment to standardization. Device independence and source code portability are assured with these powerful software tools.

GSS DRIVERS®—Provide a standard VDI interface between your PC and a wide variety of peripherals.

GSS® Bindings—The language binding provides an interface between the language source and the VDI. Bindings to Lattice C and the following MicroSoft languages are available: Fortran, Assembler, Pascal, Basic Compiler.

GSS-TOOLKITS®:

Kernel System—Subroutine library provides quality graphics for computer aided design and engineering. This toolkit adheres to standard described in Level 2b of the ANSI Graphical Kernel System (GKS) specifications.

Plotting System—Device independent subroutine library of two dimensional plotting functions for creating programmer-specified charts efficiently and easily.

Metafile Interpreter—Reads any standard metafile and converts it to graphic output commands.

Lifeboat, The Source for Software with Full Support, and Lifeboat logo are trademarks of Lifeboat Associates.
BetterBasic, Runtime System, 8087/80287 Math Module and Btrieve Interface are trademarks of Summit Software Technology, Inc.
RUN/C Lite, RUN/C Professional and RUN/C–The C Interpreter are trademarks of Age of Reason Co.
C Programming System is a trademark of Mark Williams Co.
Lattice is a registered trademark of Lattice, Inc.
GSS, GSS-TOOLKIT, and GSS DRIVERS are registered trademarks of Graphic Software Systems, Inc.

For immediate delivery or more information,
call: 1-800-847-7078 In NY: 212-860-0300

or write: **LIFEBOAT™ ASSOCIATES**
1651 Third Avenue
New York, NY 10128

LIFEBOAT™

The Source for Software with Full Support™

© 1985 Lifeboat Associates

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE SANTA CRUZ OPERATION XENIX FOR THE IBM PC

by Ronald Moak

After several weeks of anticipation, a review copy of Santa Cruz Operation's (SCO) XENIX appeared on my office desk. I had been looking forward to getting my hands a little dirty with UNIX for quite some time. Now the opportunity had arrived. Over the last year I'd read numerous books and articles and listened to a lot of people praise the virtues of UNIX and XENIX. Now I had a real chance to see what the hoopla was all about.

Having a person with no UNIX background do a review of XENIX may seem a little sacrilegious to diehard XENIX aficionados. However, I am trying to give a feeling for XENIX to those of you who feel comfortable with PC DOS, but have yet to venture into some of the more powerful operating systems.

For many people using computers, there is little reason to leave the comforting simplicity of PC DOS. But for others of us, DOS can be very limiting, especially when we're developing large, complex application programs that need to run on either single- or multi-user computers. Until recently, there has been no operating system that tied single- and multi-user machines together.

WHY UNIX/XENIX?

To give you a little perspective on my interest in UNIX/XENIX: In my work we develop a turnkey medical and dental software system. The current system is designed as a multi-user system with CPU and multiple terminals. It is an excellent system if you need a lot of terminals. However, many doctors need only a small PC and printer. Because of differences in versions of the BASIC language available on both machines, our software would need to be totally rewritten in order to run native on the IBM PC. As a turnkey system designed and supported for offices where personnel has little

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

or no interest in the computer, we needed more control over the computer system to reduce the side effects of operator error. As a result, each computer sold comes with a modem to allow us to log on to any computer to perform diagnostic work. We sell each system with the software already loaded and ready to run as soon as it is setup.

UNIX/XENIX appears, at least from all the publicity, to fill our needs for an operating system that will provide the ability to run the same software over multiple machines. The cost of having development staff deal with multiple versions of the same program, added to the cost of training the support and training personnel to deal with multiple operating systems, can quickly stagger a small company.

XENIX is Microsoft's licensed version of AT&T version III, UNIX with some of the Berkeley enhancements and some of their own. The Santa Cruz Operation did the port of UNIX to XENIX for Microsoft and is responsible (as far as I can tell) for all the other versions of XENIX. XENIX is running on a number of different machines, from Tandy's Model 16 to Altos' multi-user computers. Much to AT&T's dismay, UNIX has become a somewhat generic term relating to a host of operating systems that behave in a similar manner. There are a number of UNIX look-alikes that range widely in compatibility with true UNIX (of course there is a fair amount of incompatibility in true UNIX systems also).

I'll get into trouble if I try to explain any of the subtle differences between XENIX and UNIX. In modifying UNIX into XENIX Microsoft added some enhancements to make UNIX more palatable to business users. These enhancements include file and record locking (needed in any business program) and a new menu driven shell. The new shell is similar in operation to Microsoft's Word wordprocessing.

BRINGING UP XENIX

Two versions of XENIX are currently available for the IBM PC. Version 3.0 from SCO will run on either the PC XT or the PC AT. IBM is also selling a version of XENIX that will run on only the PC AT. The primary difference between the two is that the PC AT version is specifically designed to take advantage of the PC AT's 80286 processor. The version that I am using will run on both the PC XT or PC AT but will not support multiple users.

The PC I used for this review is one of the original 64K motherboard PCs with 448K of memory and a 10 meg hard disk. The only real change to the system was a replacement in the BIOS to allow autobooting of the computer from the hard disk. A color board with Zenith green screen composite monitor is used for video output.

While setting up the computer to run XENIX, I felt it was a good time to reorganize the hard disk. After removing all of the files from the disk and sorting out those that I could do without over the next month, I partitioned the disk to four megs of PC DOS and six megs XENIX. Somewhere deep within the bowels of the XENIX documents six megs is mentioned as the minimum usable system. What the documentation did not say is that the six megs is fine if all you want to do is run the operating system, type in a few letters, or send yourself mail. If you intend to do any serious development, you'd better allocate at least nine megs for XENIX. Similarly, if you intend to do any work using DOS, you should plan on getting one of the 20 meg hard disks.

With the six megs on my system I was able to load on most of the operating system utilities (except the communications and link floppies) and most of the development system. I ran out of disk space before everything in the development system was loaded. I spent the next couple of hours trying to decide what utilities I would not use, to make room for playing. I decided that any use of yacc or lex could be indefinitely delayed, so they were the first to go. Because I wasn't overly familiar with XENIX, it was difficult to know where to cut — I didn't know when some other program might need the files I cut. For example, I cannot get lint to work, but I don't know if I'm using the command correctly, or if I didn't load enough files on, or if I deleted too many files. The moral of the story is to start with enough storage so that you can eliminate at least one of the variables in the equation.

Aside from the limited disk space, I had other problems with bringing up XENIX. My old disk drive was not able to read all of the distribution disks. As a result, the computer issued a couple of messages equivalent to DOS's Cannot read disk message, although a little more terse. Because the installation procedures assume everything is fine, there is no indication of what to do next. In situations like this there are a number of alternatives to pursue: (a) dig through the manual and find everything there is to know about the error; (b) call the toll-free number and yell "HELP!"; (c) abort the process and try the whole thing again; (d) enter the wrong keys and end up at the login prompt. (There are probably more, but this is not a novel.) I went down path number four, not intentionally, however.

After some time exploring the XENIX file structure, I was able to locate the installation script and pick up where I left off. I still do not know what was on that disk that I did not get.

WHERE XENIX COMES INTO ITS OWN

XENIX is a rich operating system, full of many of the commands and functions that we wished were available on PC DOS. Most of the implementations of multi-tasking on PC DOS to date (including IBM's TopView) are kludges. XENIX on the IBM XT provides two different types of multi-tasking. The first is the standard submit a task to background opera-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tion from the command line. On the IBM (I do not know if this is true of the implementations for other computers) if you have sufficient RAM (384K), XENIX allows up to 10 concurrent sessions on the same console. Switching sessions is as easy as pressing Alt F1 thru Alt F10. Each session has its own login and screen. This feature is great for program development because it allows you to have source code in one screen and the program executing on the other.

If you are developing serious applications on the IBM PC, you are well aware of the quandary that has arisen in the last year for software developers. New products need to be developed using the latest in window concepts (*a la* Mac) to ease the learning curve and increase user acceptance. Large systems must be portable to reduce the cost and problems associated with proprietary operating systems and hardware. With the demand for multi-user access to software, programs that allow file and record locking need to be developed. And more and more programs will need to perform on both single-user and multi-user systems. This allows people to buy software that will allow them to migrate both the software and data from smaller to larger machines as their needs grow.

XENIX is capable of at least two of the requirements of these newer programs. It currently supports the migration path from small single-user systems to larger multi-user systems. Unlike UNIX, XENIX does support the record and file locking needed by business software. Limited window support is provided through the use of the cursor's library of functions.

THE COMPILER AND OTHER DEVELOPMENT TOOLS

For developers, the compiler has the ability to produce binary code capable of running under PC DOS. This allows one set of source code to be developed for applications running on both XENIX and PC DOS. The DOS libraries provide functions (change directory, remove directory, etc.) that you typically pay extra for on PC DOS. However, none of the large function libraries available under PC DOS is currently available; it should be fairly easy, though, to copy over and compile the non-DOS or assembler-specific source from a package such as the Greenleaf functions. Unfortunately, I was unable to test the compilation of programs on XENIX and execution on PC DOS. I did not have enough room to store the DOS libraries.

XENIX also comes with all the usual UNIX development tools. While these tools may be very powerful, their acceptance by PC DOS programmers is questionable. The complexity of the tools virtually mandates someone on the development team with a strong UNIX background. In all likelihood, only the most persistent of programmers is likely to tackle them unless someone is there to encourage their use.

IS XENIX A COMPLETE DEVELOPMENT SYSTEM?

XENIX and UNIX are often billed as the pay-one-price-get-everything development system. But as we all know, that is never the case. One of the first purchases I expect most DOS programmers to make is a good text editor. Vi may be light years beyond EDLIN, but it lacks much compared to many of the new editors available that run under PC DOS. Fortunately, a few of these new editors are finding their way up to the XENIX world. There are probably a few other items you would want, like a good source level debugger. I'm not currently aware of any, but I'm sure there either are, or will be, some available soon.

If you work a lot with both XENIX and DOS, you'll find instructions for reconfiguring the disk to boot up under PC DOS in the manuals. I found using XENIX's version of FDISK to reconfigure the hard disk's active partition a poor solution to using the DOS portion of the hard disk. I simply created a bootable DOS floppy disk with an AUTOEXEC file that logs you to Drive C when you're finished booting and resets the COMSPEC to c:\command.com. That way, if I want to use PC DOS, I boot from floppy, and if I want XENIX, I boot from the hard disk.

THE COSTS

The power of XENIX does not come without a cost. The price of XENIX (10+ times the price of PC DOS) is but a small part of the total cost of system. I read through the two operating system manuals twice before ever booting the system. You can expect to spend a significant portion of time learning how to use XENIX effectively; for example, learning which command shells will aid your productivity and which will limit it. XENIX has a large number of commands, many of which are repetitive, and all of which are cryptic.

The terseness of the commands is less a concern in program development than in the sale and acceptance of the program by the mass market. After all, how successful would Lotus be if there were no PC DOS and everyone had to use UNIX or XENIX only? I seriously doubt that half the people using Lotus would do so under XENIX. You can get Lotus up and running on a new computer in less than a half hour, without ever opening up the PC DOS manual. It takes a half hour to open the XENIX manual and find the installation instructions, let alone those for loading the operating system and loading the program that you really bought the computer to use.

As a result, most of the development on UNIX and XENIX systems is in turnkey applications where the computer comes already installed with all of the software ready to run. XENIX appears to provide an excellent medium for developing software for the massive PC market while still keeping the software compatible with the more powerful operating system.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

There is an increasing need among people who buy turnkey software to supplement the package with additional specialty packages not provided by the vendor. There is also a great need among companies developing applications to have access to the wide variety of software that cannot be effectively developed in-house, to improve the usability and saleability of their packages. For example, there is no need for an applications developer to write their own spreadsheet program to interface with their package. Because of the complexity of developing usable speadsheets, it is unlikely that any system house will write a spreadsheet module for their accounting package. With the wide acceptance of spreadsheet software, it is difficult to believe that accounting software is being written that does not directly interface.

Turnkey developers go to great lengths to shield the end user from the operating system in order to reduce support cost and ease operator use. With the use of scripts and system calls XENIX provides a number of methods to shield the operator from the headache of learning a large number of complex and cryptic commands.

Whether you should buy XENIX depends upon your needs. XENIX is certainly an attractive package in terms of cross development on the PC and XENIX/UNIX systems. It also provides another less expensive box to sell, for people with XENIX software currently running on high end systems. The IBM XT XENIX is also attractive to people with an interest in XENIX and UNIX but without the capital to buy a more powerful system. For $1300 you can buy the development package and determine for yourself whether UNIX or XENIX will have a place in product development plans. Considering the price of a couple of UNIX courses, that is a pretty cheap investment. □

THE PROGRAMMER'S SHOP™

helps compare evaluate and find products. Get answers.

SERVICE: FREE NEWSLETTER
Software development and AI on micros: trends, forecasts, controversies, innovations, and techniques. Plus announcement of 80 NEW tools. **CALL** for "Newsletter Packet."

EXPERT SYSTEMS

ExpertEASE - Inductive. PCDOS CALL

EXSYS - All RAM, Probability. Why. Trees, Solid, files. PCDOS $275

INSIGHT 1 - Probabilites, "required," menus, fast. $95. IN 2 - adds backward, forward, partitions, db2, lang. access. PCDOS $449

PROLOG

ARITY PROLOG - full, debug to ASM, C. 16 Meg, windows, strings. Compiler $1950 MSDOS $495

MPROLOG - Rich syntax, editor, segment work-space. PCDOS $725

Prolog-86 - Learn fast. Standard, tutorials, samples. MSDOS CALL

Recent Discovery

FORTRAN >> C - Fortrix creates maintainable translations. MSDOS $995

LISP

GC LISP - "Common LISP", rich, tutorial, co-routines. PCDOS CALL

IQ LISP - MacLISP and InterLISP. Well-liked, 1,000K. PCDOS $149

TLC LISP - "LISP-Machine" - like, all RAM, classes, turtle graphics, 8087. CPM-86, MSDOS $225

WALTZ LISP - "FRANZ LISP" - like, 611 digits, debug, large programs, CPM80, MSDOS $149

PROTOTYPING

Q'NIAL - Combines APL with Lisp. Full data handling. PCDOS $375

Small Talk - METHODS by DigiTalk. Full, browse, inspectors. PCDOS $250

Feature

LISP-86 — "Common" subset, tutorial, editor, PP, trace. Best to learn. All MSDOS. **Only $75**

Call for a catalog and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™
128-F Rockland Street, Hanover, MA 02339
Visa Mass. 800-442-8070 or 617-826-7531 MasterCard

New: National Accounts Division 800-446-1185

Note: All prices subject to change. Mention this ad. Some prices are specials. Ask about POs, COD. We carry ALL programming software for MSDOS, MAC, CPM, AMIGA or ST.

LEARN LISP
Write "Real" Programs with LISP-86 for $75.

- ☐ "COMMON LISP" subset
- ☐ Thorough, paced TUTORIAL
- ☐ Realistic, annotated Examples:
  - Natural Language
  - Expert System
  - File Handling
- ☐ Debugger, exit to MSDOS, "PP"
- ☐ Not "protected".
- ☐ Fast, Full Screen Editor
- ☐ 30 day Moneyback Guarantee

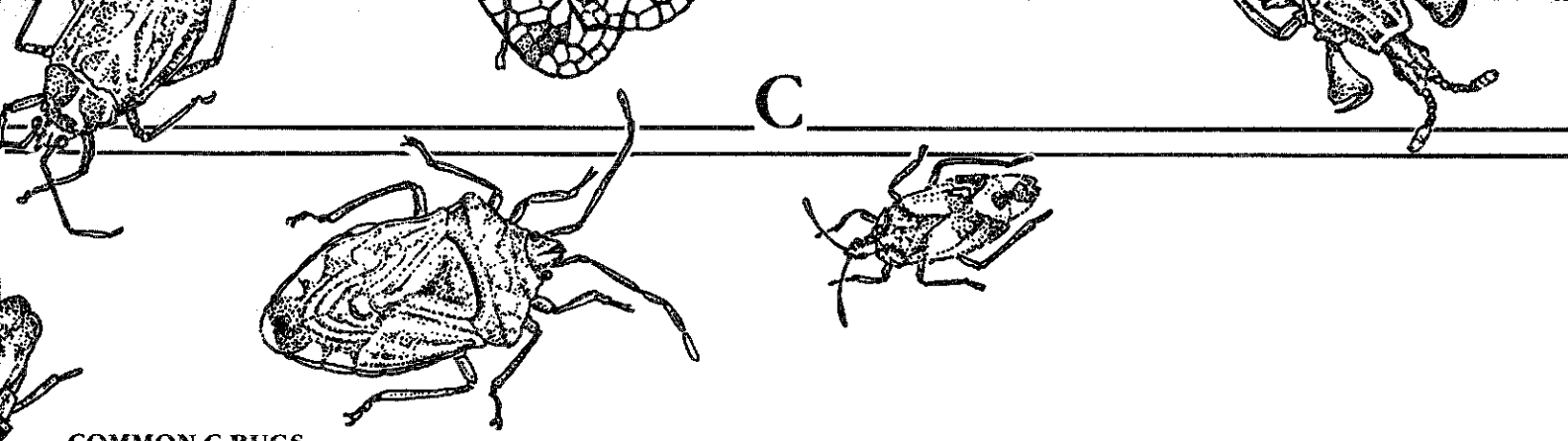
Any MSDOS or PCDOS System

Full Refund if not satisfied in 30 days.

Solution Systems

335 Washington St., Norwell, MA 02061
617-659-1571
800-821-2492

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



C

COMMON C BUGS

by Randall B. Swan

COMMON.BUG is a list of common C bugs that recur frequently in coding C. Many are traps that the professionals often fall prey to. This compendium of bugs is in no particular order. It is merely a list of situations to understand and be cognizant of. It is my hope that an understanding of these 35 or so situations will save hours of painstaking agony for those wanting to have good running code.

Common C Bugs
Randall B. Swan
The C-Team, Inc.
© May 1985
A portion of the topics in this material
is an expansion from Appendix A.9 in
Learning to Program in C
published by Plum-Hall, pg 10-13

- Uninitialized variables, especially uninitialized pointers. Auto uninitialized variables will have the junk values of the stack memory in which they're created. This leads to dangling or floating pointers where a value is placed into or retrieved from an arbitrary address. You may not notice the strange results.

- Off-by-one errors where a string or any array has advanced one address or location too far. Since there is absolutely no bounds checking in C, you'll eat into some other area of memory. For instance:

```
char s[10];
s[10] = '\0';
```

- An array always has its first element in [0], its second element in [1], and its last element in [n—1].

- Not closing out comments with an ending */ so that the next statement or even the rest of your program is eaten up by the comment. The compiler will often complain about a missing semicolon or missing closing curly brace even though that is not the problem.

- The missing semicolon. The compiler usually complains about illegal syntax in the next statement line, misleading you.

- Missing curly braces. What you think is proper indentation doesn't make a bit of difference. For instance:

```
while( .. )
        stmt1;
        stmt2;
stmt3;
```

Stmt1 is the only thing tied to the while() loop. Stmt2 isn't. Any cb program (C beatification) will display proper indentation of your code.

- A glibly typed semicolon after a while(); or for(); or if(); particularly with no space following the right parenthesis. For instance:

```
while( ... ); {
        stmt;
}
```

or

```
while( ... );
{
        stmt;
}
```

The indentation doesn't make a bit of difference here either. The compiler won't ever complain to you on this one although lint might. Here again, cb will help out.

- Break does not get you out of a block of code controlled by an if(). Break can only take you out of a for(), while(), do while(), or switch(). For instance:

```
if(...) {
    stmt1;
    if(...) break;
    stmt2;
}
stmt3;
```

The break does not get you out to stmt3; it throws you out beyond it somewhere.

- c = getchar() returning into a char instead of an int. The EOF test will always fail, and your program will sit there and never end. c must be declared as an int.

- Slash / instead of backslash \, for instance '/n' for '\n'.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

- Using = instead of = =, causing an unwanted assignment to occur inside an expression instead of evaluating properly. Assignments of non zero values will always evaluate true.

- Forgetting about the '\0' null byte in terminating strings, or clobbering it in a gone-one-too-far situation.

- Not dimensioning strings large enough to provide for the null byte or to provide for longer user input than expected.

- An improper semicolon right after the declared function name in the function body. For instance:

```
foo(a,b);
 char a,b;
{

}
```

- Confusing apostrophes (single quotes) with quotation marks (double quotes). '\0' is one byte, and "\0" is a two-byte string. This is not usually a substantial problem.

- Using getchar() to input a single character. When the user hits RETURN, another byte immediately follows as a newline '\n' often clobbering the first, or worse, affecting the next input. The best technique is to grab input as a string.

```
char stuff[81];
gets(stuff);
if( *stuff == ...)
```

  This will always preserve the first byte and turn a newline into a null byte at the end of the string.

- A similar situation occurs with scanf("%d",&n), where a number is entered followed by RETURN. The RETURN affects the next input. It's usually better this way:

```
char stuff[81];
int n;
gets(stuff);
n = atoi(stuff);
```

- Forgetting that scanf() must always have addresses.

- Using elaborate scanf() input conversion arguments. Expect anything from interactive input. Avoid scanf() if possible. Use string input and convert.

- Not declaring functions of the proper type that return pointers. In particular, forgetting declarations like

```
char *malloc();
```

  in the function that places a call to malloc().

- If a function is returning anything other than an int or char, make sure you declare it for use as the right type in the function that calls it; otherwise, the returned value will be taken as an int.

- There is absolutely no bounds checking in C. Since you can point to anywhere in memory, you can subscipt anywhere as well. Remember, num[-500] is a perfectly reasonable relative address meaning *(num-500). This would be much less obvious perhaps:

```
i = -500;
num[i] = ...
```

- There is no precedence to the postfix (after) auto increment; it ALWAYS happens after the (lvalue) expression is evaluated. For instance *p++ and *(p++) are identical. The postfix auto increment p++ will always happen last. This is one circumstance where parentheses do not force evaluation.

- The evaluation of an expression of the form

```
a[i] = i++;
```

  is an unhappy situation and is always ambiguous. Refer to page 50 K&R. It is not clear whether the auto increment occurs after the right side evaluation, after the rvalue, or after the entire statement is complete. Is it a[1] = 1; or a[2] = 1; ? You can bet it will behave differently on various compilers but always consistently with the same one. a[i++] = i; is never a problem and is perfectly consistent.

- Int's and pointers are not necessarily the same size objects. However, all pointers to any type object are always the same size.

- It's usually faster and preferable to use casts rather than unions. Unions tend to compound the obscurity of an expression. To get an int into a pointer, cast it. For instance:

```
char *p;
p = (char *) 0x100;
```

  p now points to address 100 hex, and the compiler won't complain.

- Optimizing for the wrong purpose. Optimizing will always obscure the clarity of your code. A highly ingenious C expression will not always generate ingenious object code.

  Only things that occur very frequently should be optimized. Loops and frequently called functions are the best candidates. Pull out constants (constant folding). Prioritize whether you want to optimize for time or space. Check the assembler output from the compiler to determine if efficient code is happening.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

- Static variables with initializers in functions are not initialized every time the function is called. They happen only once the first time in and are ignored on subsequent calls. This isn't a problem in main() where static is used to allow array initialization although its intent and purpose were meant to be quite different.

- The macro preprocessor does not know C and will do most of the substitutions you indicate. Arguments in a macro are not variables in C, so they have no type. Use the compiler option to generate a listing after the substitutions.
- Local variables are automatics, i.e., declared inside of functions, and are safer but slower. Global variables are externals, i.e., declared outside of functions and are faster, but can have adverse side effects. It is generally poor practice to declare all your indices globally above main() for speed considerations unless there is a particular need for speed over safety.

- Leaving out a break in a switch/case statement and falling through to other cases inadvertently.

- It is preferable to place blanks around arithmetic operators.

```
int x,a,*p;
x=a/*p;          begins a comment, it eats p & the next line
x = a / *p;      is no problem nor is  x=a/(*p);
```

- Parameter arguments in function calls are not evaluated in any particular order. The parameter list is not an expression. It may be looked at right to left or left to right, whatever the compiler writer wants to do. Most compilers will place things on the stack in reverse order, but it can vary how things go on the stack. That's not up to C. For instance:

```
foo(++bar, bar);
```

The second bar may happen before or after the auto increment.

- An idiom for an auto increment by two might be

```
++i++;
```

but is very dubious to some compilers. This will certainly work:

```
i++ + ++i;
```

but is far worse than simply

```
i++, i++;
```

- Using the logical boolean OR || instead of the bitwise OR |. If you want to OR bits together, the statement

```
x = foo || bar;
```

doesn't do it. x will have the value of 1 if either foo or bar is non zero; otherwise x is 0.

- **** STACK OVERFLOW **** is a pernicious and often impossible to diagnose bug. Nonetheless, many of these run time errors occur because there are too many local variables declared inside a function. Large array declarations, in particular those larger than 1K or 2K will do it. Stacks do not usually accommodate this quantity of storage. The solution is to reduce the internally declared variables, possibly making them external by lifting them outside the function as globals or passing them in as parameters.

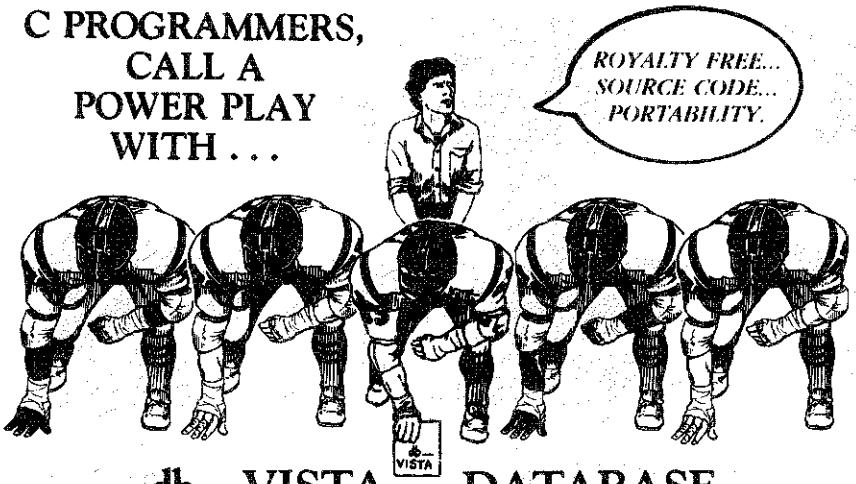
  If it is impossible to reduce the internally declared variables, then there is usually a compiler option and/or a run time option to increase the working stack size. The side effect is often slightly slower execution and additional memory consumption. It may even be handled by tweaking some magic external variable in code to your compiler such as __STACKSIZE = 4096; which will override the default. The only way to know is to dig this baby out of your compiler's reference manual somewhere.

  Another disadvantage to altering the default stack size is that deep recursion may fail and bottom out. This is a sharp point.

  Stack overflow may also occur when the wrong number or the wrong size arguments are used in function calls. Make sure the correct number of parameters exists and that the data types in the calling function match the data types in the formal parameter list. A sharp point as well. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C PROGRAMMERS, CALL A POWER PLAY WITH . . .



. . . db__VISTA DATABASE MANAGEMENT SYSTEM FOR C.

db__VISTA is a full-featured programmer's DBMS. It handles your data powerfully, yet economically without the frills that make end-user DBMS's bulky, slow and expensive to license. Use only the features you need for maximum efficiency with minimum code and effort.

Powerful lineup of features. B'-tree indexing, multiple key records, transaction processing, interactive database access utility, and file transfer utilities for dBASE, R:base and ASCII files. You even get 90 days applications development support free of charge.

Define your playbook up front. As a network model DBMS, **db__VISTA** is suited to applications development. A premium is placed on efficient use of disk storage, reduced data redundancy and fast access times allowing you to cross the goal line first.

It's your game plan. The database structure is specified by you in **db__VISTA's** data definition language (DDL). The DDL processor compiles the specification into tables (data dictionary) used by **db__VISTA's** library functions, which are called by your C program to manipulate and access the database.

db__VISTA's written in C so you can understand the signals. Source code is optional. No sweat. No royalties, either.

All this delivered for less than the price of season tickets.

Single user without source . . $195
Single user with source $495
Multi-user without source . . $495
Multi-user with source. $990
Available for most popular C compilers under **MS-DOS, XENIX,** plus most **UNIX** systems.

Go for the power play and order **db__VISTA** now. Call (206) 747-5570 or

1-800-843-3313

at the tone touch 700-992. 30 day money-back guarantee.

RAIMA CORPORATION

12201 S.E. Tenth Street
Bellevue, WA 98005 USA
Telex: 9103330300

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

YOU NEED A GOOD LIBRARY

C POWER PACKS™

COMPLETE SOURCES NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "This is a beautifully documented, incredibly comprehensive set of C Function Libraries."
— Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
— PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." — Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I (EXE files)** $99
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC. SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tools for the Programmer from Blaise Computing

Save Up To $130 On These Special Offers!

TOOLS & TOOLS 2

For C or Pascal

For a limited time, pick up both packages and save $50 off our regular list price. The C version comes with libraries for the Lattice, Computer Innovations and Microsoft (version 2.03 and 3.00) compilers. The Pascal version supports IBM and Microsoft Pascal. **$175.**

VIEW MANAGER With Source

All libraries are included. Please specify C or Pascal. Regular $425. Save $130. **$295**

Blaise Computing provides a broad range of fine programming tools for Pascal and C programmers, with libraries designed and engineered for the serious software developer. You get clearly written code that's fully commented so that it can serve both as a model and also be easily modified to grow with your changing needs. Our packages are shipped to you complete with comprehensive manuals, sample programs and source code. None of the programs are copy-protected.

FOR C AND PASCAL PROGRAMMERS:

TOOLS ◇ **$125**
Extensive string and screen handling, graphics interface and easy creation of program interfaces. Includes all source code.

TOOLS 2 ◇ **$100**
Memory management, general program control and DOS file support. Interrupt service routine support. Includes all source code.

VIEW MANAGER ◇ **$275**
General screen management. Create data entry screens that can be easily manipulated from your application program. Block mode data entry and retrieval with fast screen access.

VIEW LIBRARY Source ◇ **$150**
Source code to the VIEW MANAGER library functions.

ASYNCH MANAGER ◇ **$175**
Powerful asynchronous communications library providing interrupt driven support for the COM ports. All source code included.

FOR THE TURBO PASCAL PROGRAMMER:

Turbo POWER TOOLS ◇ **$69.95**
Extensive string support, extended screen and window management, interrupt service routines, program control and memory management, interrupt filters. All source code included.

Turbo ASYNCH ◇ **$99.95**
Interrupt driven asynchronous communication support callable from Turbo Pascal. ASYNCH is written in assembler and Turbo Pascal with all source code included.

PACKAGES FOR ALL PROGRAMMERS:

EXEC ◇ **$65**
Program chaining executive. Chain one program from another even if the programs are in different languages. Common data area can be specified. Source code included if you're a registered C TOOLS and C TOOLS 2 user.

SPARKY ◇ **$75**
Run-time resident (or stand-alone) scientific, fully programmable, reverse polish notation calculator. No limit on stack size, variables or tape. Includes all standard scientific functions and different base arithmetic.

TO ORDER, call Blaise Computing Inc. at (415) 540-5441

2034 Blake Street ◆ Berkeley, CA 94704 ◆ (415) 540-5441

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TURBO PASCAL 3.0 AND SUBDIRECTORIES

by Neil Rubenking

The code in Neil Rubenking's article can be used to access and examine directory information from within a program.

"We're three little files
who have lost our way,
B:\A:\A:\, B:\A:\A:\, B:\A:\A:\."

TURBO Pascal 3.0 brings TURBO into the world of subdirectories and full-path filenames. Read and write files on any directory, make, change, and remove subdirectories — you can even extract file handles from the File Interface Block. But file access is useless if you don't know where to look. Here is a way for programs to find any file in any subdirectory, with two useful applications.

To be sure of finding files wherever they may be, I engage a pair of detectives — DOS functions $4E and $4F. $4E is called "Find First"; give it a description of the subject and its attribute, and it returns the first file that matches. The description can include "wildcard" characters, e.g., *.COM, ??.DRV, etc. The attribute determines how DOS treats the file (see Figure 1 for a table of attributes).

"Find First" leaves its report in a mail drop — the Disk Transfer Area. By setting the DTA to the location of a known variable, I get access to the data. Not only that, I define the DTA buffer variable as the exact "shape" of the report that comes from "Find First" for quick access to details. DOS function $1A accomplishes setting the DTA.

Once "Find First" has found a file, "Find Next" fetches everything else that looks like it, based on the description that remains in the DTA.

"Find First" and "Find Next" need a coordinator. I call the boss procedure "Get_Every_File_That_Matches." The calling program supplies a template, and the boss sends out "Find First and "Find Next." First, they look for a matching file on the root directory. If there is one, they note it and find all the others. Then they look for subdirectories and repeat the whole process for each. This recursive process automatically looks at all levels of subdirectories.

A routine like this one isn't meant to be used once and forgotten — I want to use it often, under many circumstances. So, I keep it in the file GETEVERY.LIB (see Figure 2). The required types, variables, and procedures are in the package. All the calling program needs to do is $Include the file and call on "Get_Every_File_That_Matches." Well, almost all. To leave the calling program some choice of what to *do* with found files, GETEVERY.LIB contains a *forward declared* procedure called "DoSomethingWith." The calling program must supply the body of the procedure, which acts on a parameter called "found", of type "filename_type."

In many languages, the DOS function calls are not accessible, or require special Assembly Language insertions. TURBO's MSDOS procedure makes them easy. If you have the proper references, you don't have to understand registers — just copy the "regpack" type definition from the TURBO manual and use it as directed.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Fast Path for New Users of Turbo Pascal...

For TURBO PASCAL On The IBM PC

TRAINING WHEELS™

- A comprehensive **step-by-step example** to help you get started.
- **30 useful & necessary procedures**—ready to be included in your program—each with source code and with a simple program illustrating its use: Display help screens & menus quickly, check the DOS environment strings, determine where the program was loaded from, manipulate the archive & read-only bits, determine the ROM date & type of machine & type of display being used, retrieve and parse the command line arguments, make your program preserve the color attributes set for DOS, move a file to a different directory without copying it—and many more.
- Helps get your application going quicker and provides examples of how the DOS function calls are used from Turbo.
- **80-page user-guide plus diskette.**

Available for only $25.00
(plus $2 shipping). Cal. Res. add 7%.
Telephone Orders Welcomed (408) 733-4032

LANDMARK SOFTWARE
1142 Pomegranate Court
Sunnyvale, CA 94087

Take, for example, the "Set DTA" function used in this program. According to Appendix D of the DOS 2.0 manual, the function number, $1A, goes in register AH, the DTA segment goes in DS, and the DTA offset goes in DX. I use the TURBO functions "seg" and "ofs" for the addresses, but "AH" isn't on the list. It turns out that the first four registers (AX,BX, CX,DX), which hold two-byte values, can be referred to as if they were two one-byte registers — Hi and Lo. Hence, AH is the Hi byte of AX. To get a lone byte into the high byte of an integer I use TURBO's "shl" procedure — "shl 8" means SHift Left 8 bits.

Most of the DOS functions don't look at *strings* in quite the same way that TURBO does. A TURBO string consists of a length byte plus some ASCII characters. DOS uses an "ASCIIZ" string — a string of characters without a length byte, terminated by an ASCII zero ("chr(0)" or "#0" in TURBO). The "Find First" procedure wants the address of an ASCIIZ string containing the file template. No problem — I add a #0 to the string. But note that the address offset is given as "ofs(fnm) + 1;." Why "+ 1"? Because the ASCIIZ string begins *after* the length byte. If a DOS function requires the address of a buffer or an ASCIIZ string, you *must* give it the address of a *global* variable — one declared outside any procedure or function. If you try to use the address of a variable local to some procedure, you'll probably find yourself reaching for the Big Red Switch. By the same token, don't $Include GETEVERY.LIB (or any other such package) inside another procedure.

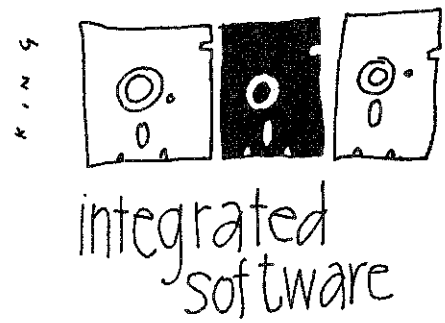
I offer a short application to show what GETEVERY.LIB can do. It's short because the $Included package does all the work. WHERE.PAS locates files matching a template. The program uses ERRMESSG.LIB to interpret any error messages returned by the function calls. Compile WHERE.PAS and use it directly from DOS. Note that the *output* of WHERE can be re-directed — "WHERE *.COM | SORT > FILELIST.DAT" will create a sorted list of all the .COM files on your hard disk.

Figure 1. — DOS file attributes

The six lowest bits of the attribute byte are significant. If a bit is 0, files with that bit set to 1 will not be chosen, but if it is set to 1, files with or without that bit will be chosen. Thus, the attribute 63 (binary 111111) could be used to select every directory entry.

bit	value	description
0	1	Read-Only
1	2	Hidden
2	4	System
3	8	Volume label
4	16	Subdirectory
5	32	Archive

The DOS system files have the attribute 7 = 1+2+4 (Read-Only, Hidden, and System). The Archive bit is set when a file is backed up and reset any time the file is changed. Thus, if a file's Archive bit is set, it has not been changed since the last backup. We want to find all files regardless of Archive status, so we set that bit when we search. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        {Make this a DOS file called GETEVERY.LIB}

type
  buffer_type   = record
                    reserved                : array[1..21] of byte;
                    attribute               : byte;
                    time, date,
                    FileSizeLo, FileSizeHi : integer;
                    name                    : array[0..12] of char;
                  end;
  filename_type = string[64];
  regpack = record ax,bx,cx,dx,bp,si,di,ds,es,flags: integer; end;

const
  Archive_bit = 32;
  SubDir_bit  = 16;

var
  file_template, filename  : filename_type; { These are GLOBAL variables. }
  registers                : regpack;       { Their addresses can be used }
  buffer                   : buffer_type;   { by DOS functions.           }
  attribute, N             : byte;

procedure DoSomethingWith(found : filename_type); FORWARD;

      {the calling program must supply the body of this procedure.
       It is FORWARD declared here so that the file-finding
       procedure can call it.}

procedure SetDTA(VAR buff);
  {Set the Disk Transfer Area to the address of the buffer}
   begin
     with registers do
       begin
         AX := $1A shl 8;     { shift the function # into AH }
         DS := seg(buff);     { set the address of the DTA to the segment }
         DX := ofs(buff);     {     and offset contained in DS:DX.        }
         MSDOS(registers);
       end;
   end;   {procedure SetDTA}

procedure Extract(VAR name : filename_type);
  {Get a filename string from the ASCIIZ string in the buffer}
   begin
     N := 0;
     name := '';
     while (N <= 12) and (buffer.name[N] <> #0) do
       begin
         name := name + buffer.name[N];
         N := N + 1;
       end;
   end;

Procedure Find_First (VAR        att : byte;
                      VAR        fnm : filename_type;
                      VAR error_code : byte);
         {find the first occurrence of a file matching the template.  If
          found, information about the file is found in the buffer.
          If not, an error code is returned.}

begin
  SetDTA(buffer);
  fnm[Length(fnm) + 1] := #0;  { make an ASCIIZ string }
  With registers do
    begin
      AX := $4E shl 8;      { shift the function # into AH }
      CX := att;            { put the file attribute in CX }
      DS := Seg(fnm);       { put the template's address in DS:DX --   }
      DX := Ofs(fnm) + 1;   {   note that the ASCIIZ string starts one }
                            {   byte AFTER the "normal" string.       }
      MsDos(registers);
      If flags and 1 = 1 then        { if the Carry Flag is set . . . }
        error_code := AX and $00FF
      else
        begin
          error_code := 0;
          att        := buffer.attribute;
          Extract(fnm);
        end;
    end;  {with}
end;

procedure Find_Next(VAR        att : byte;
                    VAR        fnm : filename_type;
                    VAR Next_error : byte);

         {find subsequent occurrences of a file matching the template.  If
          found, information about the file is found in the buffer.
          If not, an error code is returned.}

begin
  FillChar(buffer.name,12,0);
  with registers do
    begin
      AX := $4F shl 8;        { just shift the function number into AH -- }
                              {   the rest of the info is in the DTA.     }
      MSDOS(registers);
      If flags and 1 = 1 then
        Next_error := AX and $00FF
       else
         begin
           Next_error := 0;
           att        := buffer.attribute;
           Extract(Fnm);
         end;
    end;  {with}
end;


procedure Get_Every_file_that_matches(Main_template:filename_type);
var
  drive_part : string[2];
  Templ_part : string[12];

  procedure FindAll(template:filename_type);
           {FindAll is recursive.  It first finds all the matching
            files on the directory specified by the "template" parameter.
            Then it looks for subdirectories, and for each one it finds, it
            calls itself again, with that subdirectory as the template
            parameter.}

  var
    errrr, attribute : byte;
    CurrDir          : filename_type;

    procedure Read_A_Subdir;
             {The buffer and filename variables have to be "global" --
              they don't get replicated with each recursion.  So, before
              a recursive call, we have to store their values at the
              CURRENT level, and restore them afterward.}

    var
      tempBuff         : buffer_type;     { local variables, created anew }
      tempName         : filename_type;   {    at each level of recursion.}

    begin
      if (attribute and Subdir_bit = Subdir_bit) and
                      (file_template[1] <> '.') then
        begin
          tempBuff := buffer;
          tempName := file_template;
          FindAll(template + '\' + file_template);
          buffer := tempBuff;
          file_template := tempName;
        end;
     end;
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
begin
  file_template := drive_part + template + templ_part;
  CurrDir       := drive_part + template;

        {The file template is passed to the DOS functions as a
         full pathname, but it comes back as a simple filename
         plus extension, so we store the current directory in
         the variable CurrDir.}

  attribute     := Archive_bit;
  Find_First(attribute,file_template,errrr);
  if errrr = 0 then
    begin
      DoSomethingWith(CurrDir+'\'+file_template);
      while errrr = 0 do
        begin
          attribute := Archive_bit;
          find_Next(attribute,file_template,errrr);
          if errrr = 0 then DoSomethingWith(CurrDir+'\'+file_template);
        end;
    end
  else
    begin
      if errrr <> 18 then          { error #18 is "No more files."}
        writeLn(message(errrr));
    end;
  {NOW LOOK for SUBDIRECTORIES}
  file_template := drive_part + template + '\*.*';
  attribute     := Subdir_bit;
  Find_First(attribute,file_template,errrr);
  if errrr = 0 then
    begin
      Read_A_SubDir;
      while errrr = 0 do
        begin
          attribute := Subdir_bit;
          find_Next(attribute,file_template,errrr);
          if errrr = 0 then
            Read_A_SubDir;
        end;
    end;
end;

begin
  if pos(':',main_template) = 2 then
    begin
      drive_part := UpCase(main_template[1]) + ':';
      templ_part := copy(main_template,3,length(main_template)-2);
    end
  else
    begin
      GetDir(0,drive_part);
      templ_part := main_template;
      if templ_part[1] <> '\' then templ_part := '\' + templ_part;
    end;
  FindAll('');
end;
```

```
        {Make this a DOS file called Errmessg.lib}

          {This function can be used after any call to the MSDOS
           or INTR procedures that return an error message in AX.
           The signal that AX contains an error message is
           for the CARRY FLAG to be set to 1.  The test for CARRY
           FLAG set to 1 is "if Flags and 1 = 1".     }

        type
          Message_type = string[40];

        function message(code:byte):message_type;
        begin
          case code of
            1: message := 'Invalid function number.';
            2: message := 'File not found.';
            3: message := 'Path not found.';
            4: message := 'Too many open files (no handles left).';
            5: message := 'Access denied.';
            6: message := 'Invalid handle.';
            7: message := 'Memory control blocks destroyed';
            8: message := 'Insufficient memory';
            9: message := 'Invalid memory block address.';
           10: message := 'Invalid environment.';
           11: message := 'Invalid format.';
           12: message := 'Invalid access code.';
           13: message := 'Invalid data.';
           15: message := 'Invalid drive was specified.';
           16: message := 'Attempt to remove current directory.';
           17: message := 'Not same device.';
           18: message := 'No more files.';
          else message := 'Error.'
          end;
        end;


          {Make this a DOS file called Where.pas}


{$P512,D-,V-}
{ The P and D compiler directives enable redirection of OUTPUT -- for a
     list of files on your hard disk try "WHERE *.* > FILELIST."  The V-
     directive allows string routines to work on variables of differing
     lengths.}


    {This program uses the routines in "GETEVERY.LIB" to find all files
     in whatever directories matching the template you enter on the
     command line.  The ParamCount and ParamStr routines are part of
     TURBO 3.0 -- DO upgrade!)  }


program Where_is_it;
type
  template_type = string[14];
{$I errmessg.lib}
{$I getevery.lib}

  {This procedure was FORWARD declared in GETEVERY.LIB}
  procedure DoSomethingWith;{(found : filename_type);}
    begin
      writeLn(found);
    end;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE 11TH WEST COAST
COMPUTER FAIRE

Decade II

the odyssey continues . . .

APRIL 3–6, 1986
MOSCONE CENTER
SAN FRANCISCO

- **Thousands of Computer Products** exhibited by hundreds of leading and innovative companies, including IBM, Apple, and many others.
- **Special Areas For "Hands-On" Demos.**
- **A Comprehensive Conference Program** (included in admission) attracting industry speakers and computer users from business, education, government and scientific, engineering & research communities.
- **Professional Development Seminars.**
- **Free One-On-One Consultations.**
- **Incredible "Show Pricing"** on all kinds of products.

Send in for more information:

Please send me information on attendee preregistration discounts.

Name _____

Company _____

Street _____

City/Town _____ State ____ Zip _____

Phone _____

I would like to ☐ Attend ☐ Exhibit ☐ Present A Paper

Send this coupon to:
The 11th West Coast Computer Faire
181 Wells Avenue
Newton, MA 02159
You can call us at **(617) 965-8350.**

PJ

An exclusive production of **Computer Faire, Inc.**/A Prentice-Hall Company

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dBASE II Routines

By Brett Salter

This second group of dBASE II routines from Brett Salter of Data Base Decisions includes LP2 for lightpen/keyboard input, READNO for reading a numeric memvar, TIME for getting time from DOS, DOSBIOS to perform DOS/BIOS functions, ANDOR to and/or/xor two bytes, and KEYREAD for reading the keyboard status.

```
csegsegment para public 'code'
org100h

; LP2 for ISN
; light pen/keyboard input

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value

dsubproc near
jmp short p010; start point

a000equ this word
devicedb 0; device used (+2)
rowdb 0; row if lightpen (+3)
columndb 0; column if lightpen (+4)

p010:call p020; get address
a020equ this byte
p020:pop si
sub si,a020-a000; get address of data
push si ; save address for end

mov cl,[bx]; get length of string
mov ch,0
inc bx; point to first avail char

p025:mov ah,1; empty keyboard buffer
int 16h
jz p030 ; no char found

mov ah,0
int 16h ; get the char
jmp p025

p030:mov ah,4; try light pen
push bx
push cx
int 10h
pop cx
pop bx
cmp ah,1; pressed?
jnz p040; no

mov al,2; setup device indicator
jmp p100; and quit

p040:mov ah,1; try keyboard
int 16h
jz p030 ; nothing ready to read

mov ah,0
int 16h ; get char in al

cmp al,0; double scan code?
jz p030 ; yes

p045:cmp al,27; escape?
jnz p050; no

mov al,0; yes - set device indicator
jmp p100; and quit

p050:cmp al,8; backspace?
jnz p060; no

cmp ch,0; in first position?
jz p030 ; yes - no can do

dec ch
dec bx
call p200; move back one
mov al,' '
mov [bx],al; space out string
call p200; display space
mov al,8
call p200; and move back again
jmp p030

p060:cmp al,0dh; carriage return?
jnz p070; no

mov al,1; yes - set device indicator
jmp p100; and quit

p070:cmp ch,cl; room left in buffer?
jb p075 ; yes
mov al,7; no - beep
call p200
jmp p030

p075:call p200; display char in al
inc ch; add to position count
mov [bx],al; add char to buffer
inc bx
jmp p030; and back to top

p100:pop si; get address back
mov [si+1],dh; save row for light pen
mov [si+2],dl; save column for light pen
mov [si],al; save device indicator
ret; return to dBASE

p200proc near; display character in al
push ax ; save character
push bx
mov ah,14; display it
xor bx,bx
int 10h
pop bx
pop ax
ret
p200endp

dsubendp
csegends
enddsub


csegsegment para public 'code'
org100h

; readno
; read a numeric memvar

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value
memnamequ 09a00h; memory variable table

dsubproc near
jmp short p010; start point

a000equ this word
numberdw 0; the number read (+0)
vnamedb 'NUM',0                ; name of the numeric field

p010:call p020; get address
a020equ this byte
p020:pop di
sub di,a020-a000; get address of data

mov cx,64
mov si,memnam; point to memory variable table

p025:mov ax,[si]; get mem var name
cmp ax,[di+2]
jnz p030; not equal
mov ax,[si+2]; check next word
cmp ax,[di+4]; equal
jz p040 ; yes - found num
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
p030:add si,16; point to next item
loop p025
jmp p080; no find - error

p040:mov al,[si+11]; number?
cmp al,'N'+128
jnz p080; no

mov si,[si+13]; address of number
mov al,[si]; get decimal position
sub al,3fh
jb p080 ; fractional
mov ch,al; no of items

xor ax,ax; total
mov bh,0

p050:inc si; point to next byte
mov bl,[si]; get next byte
mov cl,4
shr bl,cl; high part in bl
call p200; multiply and add
dec ch
jz p060 ; done

mov bl,[si]
mov cl,4
shl bl,cl
shr bl,cl; low part in bl
call p200; multiply and add
dec ch
jz p060 ; done
jmp p050

p060:mov [di],ax
jmp p100

p080:xor ax,ax
mov [di],ax

p100:ret; return to dBASE

p200proc near; multiply and add
push cx
mov cx,10
mul cx
pop cx
add ax,bx; add to total
ret
p200endp

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; time
; get time from dos

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0df00h; sp value

dsubproc near
mov al,[bx]; get length of arg
cmp al,8; is it 8?
jnz p100; no - return
mov ah,2ch
int 21h ; get time
mov al,ch; hours
call p120
mov [bx+1],ax
mov ch,':'
mov [bx+3],ch; colon
mov al,cl; minutes
call p120
mov [bx+4],ax
mov [bx+6],ch; colon
mov al,dh; seconds
call p120
mov [bx+7],ax
p100:ret; return to dBASE

p120proc near; convert time
aam
add ax,3030h
xchg ah,al
ret
p120endp

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; dosbios
; perform dos/bios functions

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0df00h; sp value

dsubproc near
jmp short p010; start point

datumequ this word
raxdw 0; ax
rbxdw 0; bx
rcxdw 0; cx
rdxdw 0; dx
intnodb 0; interrupt

p010:call p020; get address
offerequ this byte
p020:pop si
sub si,offer-datum; get address of data

; set registers for call
mov al,[si+8]; interrupt in al
call p030
p030:
pop bx       ; ip in bx
a030equ this byte
add bx,a040-a030
mov [bx],al       ; set interrupt
mov ax,[si]
mov bx,[si+2]
mov cx,[si+4]
mov dx,[si+6]

push si
int 00h ; overlaid by intno%
a040equ this byte

pop si
mov [si],ax
mov [si+2],bx
mov [si+4],cx
mov [si+6],dx

p100:ret; return to dBASE

dsubendp
csegends
enddsub



csegsegment para public 'code'
org100h

; andor
; and/or/xor two bytes

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0dfffh; sp value

dsubproc near
jmp short p010; start point

a000equ this word
operdb 0; operation (+0)
byteldb 0; first byte (+1)
byte2db 0; second byte (+2)
byte3db 0; result (+3)

p010:call p020; get address
a020equ this byte
p020:pop si
sub si,a020-a000; get address of data

mov ax,[si+1]; get arguments in ax
mov bl,[si]; get operation
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Programmers Quit Working So Hard!!

THE GREENLEAF FUNCTIONS™

The GREENLEAF FUNCTIONS GENERAL LIBRARY has over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and systems interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library.**) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you.

THE GREENLEAF FUNCTIONS™

The Library of C Functions that probably has just what you need . . . **TODAY!**

- already has what you're working to re-invent
- already has over 200 functions for the IBM PC, XT, AT, and compatibles
- already complete . . . already tested . . . on the shelf
- already has demo programs and source code
- already compatible with all popular compilers
- already supports all memory models, DOS 1.1, 2.0, 2.1
- already optimized (parts in assembler) for speed and density
- already in use by thousands of customers worldwide
- already available from stock (your dealer probably has it)
- It's called the **GREENLEAF FUNCTIONS.**

The Library of C Functions is Waiting for You

Specify compiler when ordering. Add $7.00 for UPS second-day air (or $5.00 for ground). Texas residents add sales tax. Mastercard, VISA, check or P.O. In stock, shipped same day.

General Libraries	$185
Comm Library	$185
CI C86 Compiler	$349
Lattice C	$395
Mark Williams	$475

For Information: 214-446-8641

Prices are subject to change without notice.

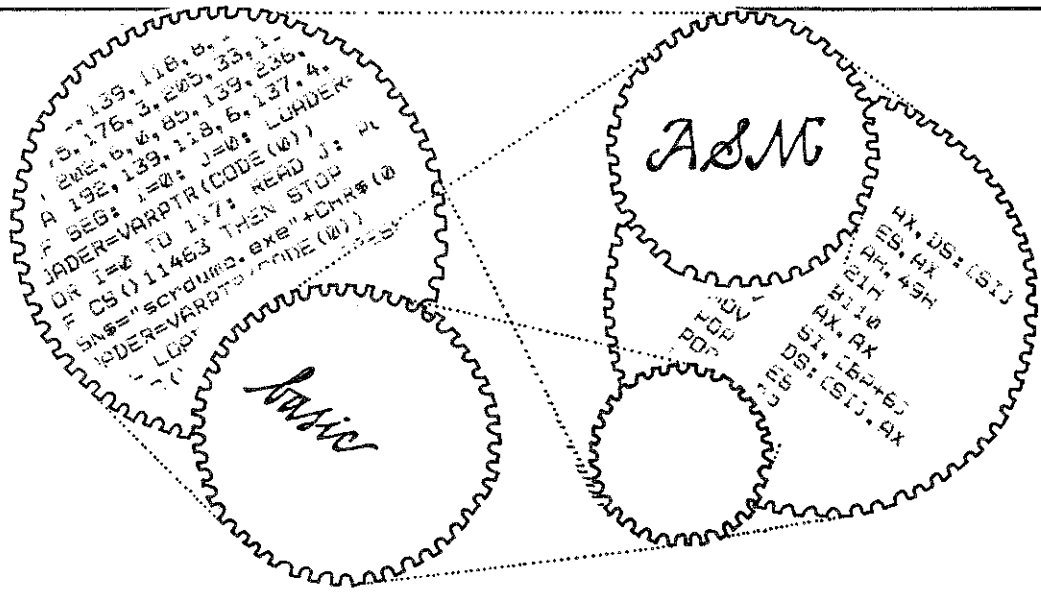
GREENLEAF SOFTWARE
2101 HICKORY DR.
CARROLLTON, TX 75006


```
cmp bl,'a'              ; upper case?
jb p030 ; no

sub bl,'a'-'A'          ; convert to upper case

p030:cmp bl,'A'             ; and?
jz p040 ; yes

cmp bl,'O'              ; or?
jz p050 ; yes

cmp bl,'X'              ; xor?
jz p060 ; yes

mov bl,'?'              ; unknown
mov [si],al
jmp p100
```

```
p040:and ah,al; and
jmp p090

p050:or ah,al; or
jmp p090

p060:xor ah,al; xor
jmp p090

p090:mov [si+3],ah

p100:ret; return to dBASE

dsubendp
csegends
enddsub


csegsegment para public 'code'
org100h

; keyread
; read keyboard status

assume cs:cseg,ds:cseg,ss:nothing,es:nothing

spvalequ 0df00h; sp value

dsubproc near
mov al,[bx]; get length of arg
cmp al,9; is it 9?
jnz p100; no - return
push es
xor ax,ax
mov es,ax
mov di,417h; keyboard flag
mov ah,es:[di]; get keyboard flag 1 in ah
pop es
mov al,[bx+1]; get function
mov cx,8; loop counter
mov si,bx
inc si
inc si; 2nd byte of argument
cmp al,'R'              ; read?
jz p010 ; yes
cmp al,'C'              ; change?
jz p050 ; yes
jmp p100; no - return

p010:; read keyboard
mov al,80h; starting check
mov dh,'0'              ; off
mov dl,'1'              ; on

p020:test ah,al; is the bit on?
jnz p030; yes
mov [si],dh; no - turn it off
jmp p040

p030:mov [si],dl; turn it on

p040:inc si
shr al,1
loop p020; done?
jmp p100

p050:; change keyboard
mov ah,0; all kb flags off
mov al,80h; starting check

p060:mov dl,[si]; get byte
cmp dl,'1'              ; turn it on on?
jnz p070; no
or ah,al; yes

p070:inc si
shr al,1
loop p060; done?

push es
xor bx,bx
mov es,bx
mov di,417h; keyboard flag
mov es:[di],ah; change keyboard flag 1
pop es

p100:ret; return to dBASE

dsubendp
csegends
enddsub
```

□

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



A BETTER WAY TO INTERFACE BASIC AND ASSEMBLER-

by John W. Jackson

The most flexible method for calling assembler subroutines from the BASIC interpreter is to make the subroutine a resident extension of DOS. However, this method has a serious drawback. How do you get rid of the routine when you are done with it? It would be nice to reclaim this memory for other uses without having to reboot the system, particularly if the subroutine is large. The other, non-resident methods all have drawbacks such as not allowing relocatable expressions or not working in conjunction with the SHELL command. Why can't we have our cake and eat it too? This article describes a method that has all of the flexibility and features of resident subroutines, plus a few more, and allows your BASIC program to control the dynamic loading and unloading of assembler routines. All memory used by the subroutines will be automatically freed when BASIC terminates. This method also allows a great deal of flexibility in controlling memory usage. You may load a subroutine, use it, and when it is no longer needed, unload it and reuse that memory for another subroutine.

HOW DOES IT WORK? The method is really fairly simple. DOS already provides the functions necessary to do the task. These are the memory allocate function 48H, the release memory function 49H, and the little known or used loader function of 4BH. These functions are combined to form a small and simple loader routine. The routine is passed the filename of the subroutine we wish to load. The loader routine uses 48H to acquire memory to load the subroutine into. The routine is then loaded using 4BH subfunction 3. The segment address of where the subroutine was loaded is returned to the calling BASIC program. The BASIC program may then address and call the subroutine by use of DEF SEG. If desired, the subroutine may be unloaded by calling the UNLOAD entry point of the loader routine. The address is passed to the UNLOAD routine, which then frees the memory using the 49H function.

AN EXAMPLE

The source of a sample BASIC program is shown in Listing 1. The assembler source for a simple subroutine that we are going to call from our BASIC program is shown in Listing 2. This subroutine will invoke the screen print function and cause a copy of the current screen to be printed. The assembler source for the loader routine is shown in Listing 3.

Let's look at the BASIC source first. The assembler loader routine has been converted to DATA statements and will be poked into the integer array CODE. The subroutine we want to call is contained in the file SCRDUMP.EXE. In line 220, this filename is placed in a BASIC string variable. Note that a binary zero is appended to the string. This puts the filename in what's known as ASCIIZ format. The loader function 4BH likes it that way, and it's easier to do in BASIC than to copy the string and append the zero in the loader routine. Line 240 calls the loader routine. Assuming the loader was successful, the segment address of the loaded subroutine will be in the BASIC variable ADDRESS. Line 260 addresses the subroutine by issuing a DEF SEG, and in line 270 the subroutine is actually called. We may, of course, call SCRDUMP as many times as we like without having to reload it again. Line 300 calls the UNLOAD entry point of the loader routine and frees the storage used by the SCRDUMP routine. This step is not necessary unless the storage is required by another subroutine or by the SHELL command. Once the SCRDUMP routine has been unloaded, it must be loaded before it can be called again.

Now let's examine the assembly source of the SCRDUMP routine. This subroutine is coded just as you would code any BASIC assembly language routine — with one exception. The routine should free any memory that it is not going to use. When the loader obtains memory for the subroutine, it allocates all of available memory to it. The subroutine should therefore be a good citizen and free any memory that

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Introducing Periscope II Professional Debugger and Break-out Switch

New Periscope II includes a remote break-out switch that does not need its own slot!

(Periscope is) "the best value in development tools currently on the market . . . the most essential element of my 'developer's toolbox'." —Jeff Garbers

"Very powerful for debugging and teching. Better than Afros by far." —Wynn Bailey

The break-out switch ***"really sets Periscope apart from the typical software-only debuggers."*** Hung system or locked keyboard? Press the switch to get control!

Periscope's symbol support ***"beats the daylights out of snooping through a map file and making notes".*** See high-level line numbers and source code, too!

"Feel right at home" **in no time** with commands that logically extend Debug's!

Periscope's speed makes other debuggers ***"look absolutely sluggish"***! It's written entirely in assembler and uses DOS only for file access.

Has all the standard features plus:

- Debug with over 75 breakpoint options
- New! Write your own breakpoint tests
- New! Traceback
- New! Do in-line symbolic assembly
- Debug using one or two monitors
- Recall command lines
- New! Debug with high-level source code
- New! Redefine windows while debugging
- New! View text files while debugging
- Debug device drivers, non-DOS and memory-resident programs
- New! Customize Periscope via user exits
- New! Display 8087/80287 status
- New! Use Periscope with an EGA

Periscope requires: IBM PC, XT, AT, or close compatible; DOS 2.0 & later; 128K RAM; 1 Disk Drive; 80-column Monitor.

Periscope II, break-out switch, manual, reference card and software . . . $145!

Periscope I also includes the write-protected RAM board to protect crucial debugger code. It's just $295!

The US Navy gets Periscopes from us . . . shouldn't you? Order today!

Order/Information Call Toll-Free:

800-722-7006

30-Day Money-Back Guarantee

Data Base Decisions • 404/256-3860
14 Bonnie Lane • Atlanta, GA 30328

it is not going to use. If you are not going to load any other subroutines and if your BASIC program does not use the SHELL command, then you can get away with omitting this step.

In summary, this method has the following features:

- Your assembly language routines may contain relocatable expressions.
- The subroutines may be as large as memory permits
- No memory is taken from BASIC's 64K workspace.
- This scheme works properly with BASIC programs that use the SHELL command.
- No excess memory is left allocated or resident after BASIC terminates.
- Your BASIC program may create dynamic overlays by controlling the loading and unloading of subroutines as needed.

For a diskette containing the source code used in this article, send $5.00 to

BASIC LOADER
Jackson Software
1729 Mayflower Dr.
Carrollton, TX 75007
214-492-1982 □

Listing 1 ...

```
100 DEFINT A-Z
110 DIM CODE(58)
120 DATA 235,8,144,235,83,144,0,0,0,0,85,139,236,6,187,255,255,180,72,205,33
130 DATA 129,235,128,0,127,3,235,46,144,180,72,205,33,114,42,46,163,6,0,46,163
140 DATA 8,0,139,118,8,137,4,139,118,10,139,84,1,140,200,142,192,187,6,0,180
150 DATA 75,176,3,205,33,114,8,43,192,235,4,144,184,4,0,139,118,6,137,4,7,93
160 DATA 202,6,0,85,139,236,6,139,118,8,139,4,142,192,180,73,205,33,114,2,43
170 DATA 192,139,118,6,137,4,7,93,202,4,0,0
180 DEF SEG: I=0: J=0: LOADER=0: UNLOAD=0: ADDRESS=0: RC=0: CS=0
190 LOADER=VARPTR(CODE(0))
200 FOR I=0 TO 117: READ J: POKE LOADER+I,J: CS=CS+J: NEXT
210 IF CS<>11463 THEN STOP                  '* make sure loader code is correct
220 DSN$="scrdump.exe"+CHR$(0)              '* make routine name in ASCIIZ format
230 LOADER=VARPTR(CODE(0))
240 CALL LOADER(DSN$,ADDRESS,RC)            '* load the routine
250 IF RC<>0 THEN STOP                      '* check return code
260 DEF SEG=ADDRESS: SCRDUMP=0              '* address the loaded routine
270 CALL SCRDUMP                            '* call the loaded routine
280 DEF SEG                                 '* address BASIC's data segment
290 UNLOAD=VARPTR(CODE(0))+3
300 CALL UNLOAD(ADDRESS,RC)                 '* unload the routine
310 IF RC<>0 THEN STOP                      '* check return code
```

Listing 2 ...

```
;** TODSTAMP 09-22-1985 19:58:03
;*****************************************************************
;*****************************************************************
;**
;**   CALL SCRDUMP
;**
;*****************************************************************
;*****************************************************************
CSEG    SEGMENT PARA PUBLIC 'CODE'
SCRDUMP PROC  FAR
        ASSUME CS:CSEG
        PUBLIC SCRDUMP

        JMP   A100

FLAG    DB    0

A100:   CMP   FLAG,1        ;IS THIS FIRST TIME
        JE    A200          ;NO
;**
;**  RELEASE ANY MEMORY WE ARE NOT GOING TO USE
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;**
        PUSH  ES
        MOV   AX,CS
        MOV   ES,AX
        MOV   BX,OFFSET THEEND
        ADD   BX,15
        MOV   CL,4
        SHR   BX,CL
        MOV   AH,4AH
        INT   21H             ;CALL SETBLOCK TO RELEASE MEMORY
        MOV   FLAG,1
        POP   ES

A200:   INT   5               ;PRINT THE SCREEN CONTENTS
        RET
THEEND  EQU   THIS BYTE
SCRDUMP ENDP
CSEG    ENDS
        END
```

Listing 3 ...

```
;** TODSTAMP 09-24-1985 19:21:41
;*****************************************************************************
;*****************************************************************************
;**
;**   CALL LOADER(DSN$,ADDRESS%,RC%)
;**
;**   A SUBROUTINE TO DYNAMICALLY LOAD ASSEMBLY LANGUAGE SUBROUTINES
;**   FOR THE BASIC INTERPRETER
;**
;**   INPUTS:
;**     DSN$    - A BASIC STRING CONTAINING THE NAME OF THE FILE TO BE LOADED
;**               IT SHOULD BE COMPLETE WITH DRIVE AND PATH IDENTIFIERS IF
;**               REQUIRED AND THE FILE EXTENSION
;**               IT MUST BE TERMINATED BY A BYTE OF ZEROS SO IT WILL BE IN
;**               ASCIIZ FORMAT FOR THE LOADER
;**
;**   OUTPUTS:
;**   ADDRESS% - AN INTEGER CONTAINING THE SEGMENT ADDRESS OF WHERE THE
;**               ROUTINE WAS LOADED
;**
;**    RC%      - AN INTEGER CONTAINING THE RETURN CODE
;**               A VALUE OF ZERO INDICATES SUCCESS
;**
;*****************************************************************************
;*****************************************************************************
CSEG    SEGMENT PARA PUBLIC 'CODE'
LOADER  PROC  FAR
        ASSUME CS:CSEG
        PUBLIC LOADER

        JMP   A100            ;ENTRY POINT FOR LOADER
        JMP   B100            ;ENTRY POINT FOR UNLOAD

P1      DW    0               ;PARAMETER BLOCK FOR 4B FUNCTION
P2      DW    0

A100:   PUSH  BP
        MOV   BP,SP
        PUSH  ES

        MOV   BX,0FFFFH       ;ALLOCATE ALL OF AVAILABLE MEMORY
        MOV   AH,48H
        INT   21H
        SUB   BX,80H          ;SAVE 2K FOR SYSTEM LOADER
        JC    A110
        JMP   RET4            ;NOT ENOUGH MEMORY

A110:   MOV   AH,48H
        INT   21H
        JC    RET

        MOV   P1,AX           ;ADDRESS OF WHERE TO LOAD ROUTINE
        MOV   P2,AX
        MOV   SI,[BP+8]
        MOV   DS:[SI],AX      ;SET ADDRESS FOR CALLER

        MOV   SI,[BP+10]
        MOV   DX,DS:[SI+1]    ;DS:[DX] -> ASCIIZ FILENAME TO LOAD
        MOV   AX,CS
        MOV   ES,AX
        MOV   BX,OFFSET P1    ;PARM BLOCK FOR LOAD FUNCTION
        MOV   AH,4BH
        MOV   AL,3
        INT   21H             ;CALL THE LOADER
        JC    RET
        SUB   AX,AX
        JMP   RET

RET4:   MOV   AX,4

RET:    MOV   SI,[BP+6]
        MOV   DS:[SI],AX      ;SET CALLERS RETURN CODE
        POP   ES
        POP   BP
        RET   6
```


MBASIC Programmers

Save **Time & Money** with

HELP MESSAGE SYSTEM™

Minimize documentation! Create and edit, context sensitive, interactive help and/or tutorial functions. Turns your text information into callable libraries. Overhead to your program is insignificant.

B-PLUS TREE SUB-SYSTEM™

Can use unique or non unique keys. Direct and sequential accessing, variable length keys, multiple non unique keys and and multiple keys per record. Get First, Next, Last or Prior key in sequence. Can access a key field in a file of 32,000 records in less than 1 second!

SCREEN BUILDER SYSTEM™

Using either the built in editor or your word processor, create edit fields of any length anywhere on the screen with full screen editing within each field and full screen movement from field to field. The generated code is ready to insert in your program. Does not make any calls to BIOS. Does not use BLOAD or BSAVE. Bulletproof editing.

EZ SCREEN™

An Assembler language routine embedded in a basic subroutine to perform screen handling and windowing functions when called from a Basic program.

Priced As Low As **$59** Each

30 DAY $ BACK GUARANTEE
NOT COPY-PROTECTED
NO ROYALTY

Any 1 Program	$79	
Any 2 Programs	$138	($69 each)
Any 3 Programs	$198	($66 each)
All 4 Programs	$236	($59 each)

Visa, Mastercard, Checks or C.O.D.
CA residents add 6.5% sales tax
No shipping charges

STERLING CASTLE

702 Washington Street Suite 174
Marina Del Rey, California 90291
(213)306-3020

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GROWING OLD?

...waiting for C programs to compile and link?

Use C-terp
the complete C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor-- no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, etc. etc. For the ultimate in customization, editor source is available for a slight additional charge of $98.00.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Object Module Support-- Access functions and externals in object modules produced by your compiler. **New:** We are now supporting **Microsoft** 3.0, **Mark Williams** & **Aztec C** in addition to C.I. C86 & Lattice.
- Complete Multiple Module Support.
- Symbolic Debugging-- Set breakpoints, single-step, and directly execute C expressions.
- Many more features including batch mode and 8087 support.
- **Price: $300.00 (Demo $45.00) MC, VISA**

Price of demo includes documentation & shipping within U.S. PA residents add 6% sales tax. Specify compiler.

- C-terp runs on the IBM PC (or compatible) under DOS 2.x with a suggested minimum of 256Kb of memory. It can use all the memory available.

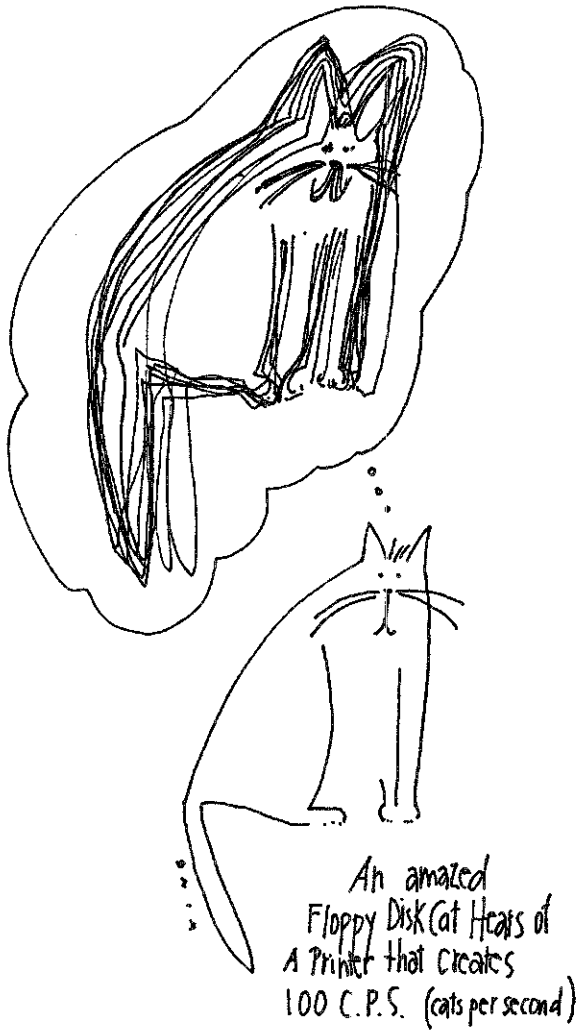
GIMPEL SOFTWARE

3207 Hogarth Lane • Collegeville, PA 19426

(215) 584-4261

*Trademarks: C86 (Computer Innovations), Lattice (Lattice Inc.) IBM (IBM Corp.), C-terp (Gimpel Software), Microsoft (Microsoft), Aztec (Manx)

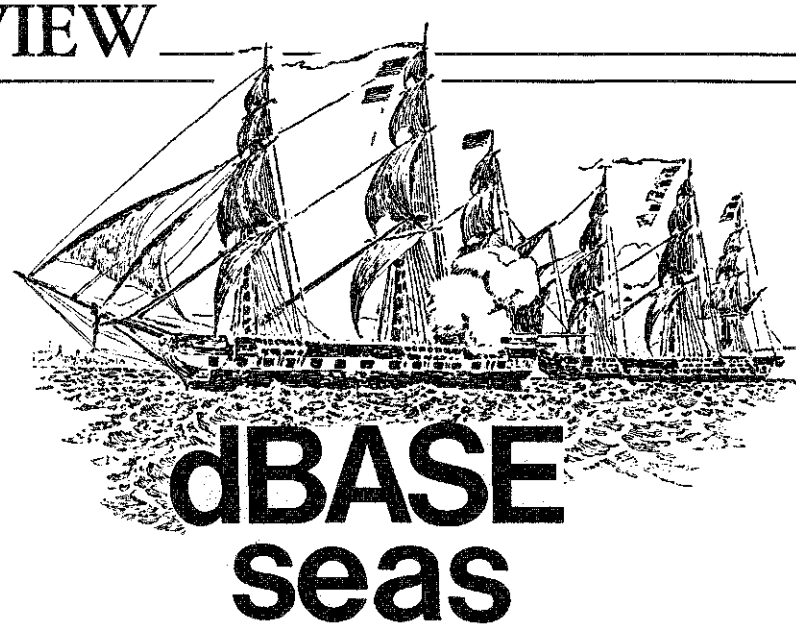

```
;*****************************************************************************
;*****************************************************************************
;**
;**    CALL UNLOAD(ADDRESSZ,RCZ)
;**
;**     FREES MEMORY USED BY PREVIOUSLY LOADED ROUTINE
;**
;**     INPUTS:
;**
;**      ADDRESSZ - AN INTEGER CONTAINING THE ADDRESS OF THE LOADED ROUTINE
;**
;**     OUTPUTS:
;**
;**       RCZ     - AN INTEGER CONTAINING THE RETURN CODE
;**                 A VALUE OF ZERO INDICATES SUCCESS
;**
;*****************************************************************************
;*****************************************************************************
B100:     PUSH   BP
          MOV    BP,SP
          PUSH   ES
          MOV    SI,[BP+8]

          MOV    AX,DS:[SI]     ;THE LOAD ADDRESS
          MOV    ES,AX
          MOV    AH,49H
          INT    21H            ;FREE MEMORY
          JC     B110
          SUB    AX,AX
B110:     MOV    SI,[BP+6]
          MOV    DS:[SI],AX     ;SET CALLERS RETURN CODE
          POP    ES
          POP    BP
          RET    4

LOADER    ENDP
CSEG      ENDS
          END
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CLIPPER SAILS ON dBASE SEAS!

by Paul Barkley



One of the great things about capitalism is that if you don't like the way something is done, you're free to go out and do it a different way — perhaps making your fortune in the process. dBASE II and III have become very much a staple item for many programmers, and there are few who would ever go back to writing file handling routines in a high level language, but there are many who think they could do a better job of designing dBASE II or III. Most competitors come up with programs that aren't very similar, and many of these companies have found the market place not particularly interested. In a moment I'll talk about a company that decided to leave the core of dBASE III alone — merely extending it — and to turn it into a compiler in the process.

One of the reasons for dBASE II's success has been the fact that it comes with a procedural language. In my opinion, most users have no business fooling around with data bases that require direct manipulation. That has been a problem with Condor, Rbase, and dBASE II's dot prompts — suitable only for very experienced users and those who are constantly changing data base tasks.

dBASE II, though, left quite a bit to be desired. The indexes were easy to scramble, only two files could be open at a time, and some handy features were missing. When dBASE III came out, it was like the difference between night and day. I can't imagine anyone using dBASE II unless they're getting paid a lot of money to maintain some old code, or have a masochistic streak.

dBASE III, though, has a few problems, such as the mysterious garbage added to keys that are not a multiple of 4 bytes and other subtle quirks that appear only when deadlines and tempers are short. For someone writing applications code, though, dBASE III has three glaring defects:

1. It's copy protected, and it's a real nuisance to work with the multiple disks.

2. Programs are still interpreted, and although they run pretty well, they could run a whole lot better if they were compiled.

3. Program source code is completely vulnerable to inspection and tampering by users.

In addition, Ashton-Tate has managed to alienate a large number of people with their policies on bug fixes, updates, copy protection, etc. Although many of the things they do are really not that terrible in themselves, these policies in the aggregate, coupled with Ashton-Tate's success, have given them a bad reputation in many quarters. Apparently, the only thing many people hate worse than a sore loser is a poor winner. Ashton-Tate seems to be trying to overcome some of these problems — they even publish *Tech Notes* now that explain a lot of things. (Of course they refer to "bugs" as "anomolies" — I guess that's better than calling a bug an "intentional design specification.")

Time, tide, and competition wait for no man, woman, or corporation, and in the best capitalistic tradition a start-up company called Nantucket decided that they could do things better than Ashton-Tate — the time was right for a dBASE III compiler. I'll leave the esoteric arguments about compiler implementations to those who specialize in that sort of work — I want to talk about the incredible difference it makes when you switch from using dBASE III to using Clipper and some of the practical problems you encounter.

One of the most important considerations is that Clipper has absolutely nothing to do with dBASE II or III, other than to compile programs that look almost exactly like dBASE II or dBASE III programs. You don't need dBASE III to write the programs, and the user doesn't need dBASE III to run the programs either. In fact, the user doesn't need any run-time module at all — Clipper produces an .EXE file that can be run directly from DOS.

This means that if you're planning on distributing a product, the end user can save $700 by not having to buy dBASE. This can make a dramatic difference in how your product is accepted, particularly if it's low priced to begin with. Ashton-Tate obviously hates this.

On the other hand, if you're going to write Clipper programs, you'd better own a legitimate copy of dBASE III because you *might* need the interpreter and you're *certainly* going to need the manual — the Clipper manual assumes you're a dBASE III expert and covers only topics related to going from an in-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

terpreted environment to a compiled environment. If you're not proficient in dBASE III, you'd better plan on spending some time testing pieces of code in the interpreted language — otherwise, you'll spend a lot of time compiling stuff that doesn't run because it isn't good dBASE III code.

The Clipper manual comes boxed like the dBASE III manual and shares the same defect — it's a little too tall to fit on a shelf with DOS-sized manuals. There are only 60 pages, divided into three chapters and an appendix, plus another 20 typewritten pages explaining things that were left out of the other 60 pages. It does come with a nice homey apology for the manual, though, that I find a refreshing change from more traditional polished works. After working with Clipper for a while, go back and take a look at the dBASE III manual — you'll see where quite a bit of Ashton-Tate's budget went.

Clipper, unfortunately, is copy protected and must be installed on your hard disk. A simple "CLIPCOPY C:H" takes care of the situation, and after an inordinate amount of grinding on the floppy drive and some prompted disk switching, about 400K of information gets transferred, including a linker called PLink86 from Phoenix Software Associates, Ltd. If you wish, you can use your DOS linker instead.

When I first used Clipper, I was somewhat overwhelmed by the differences between interpreted dBASE and compiled Clipper programs — even the skimpy 60 pages of the manual managed to raise quite a few issues. I decided that the best way to understand the product would be to simply compile and link an existing dBASE III program — preferably something very simple.

I chose a simple phone list manager that I had written — nothing to it. To compile and link in one command, you can use the built in batch file. I typed

C> CL PHONES

and sat back to watch. It took about 20 seconds to compile my original dBASE III source file (written using WordStar) and another three minutes to link. The original .PRG file was 2,560 bytes, but eventually the resulting .EXE file was 118,928 bytes!

Several points became apparent at once. The compiler generated a polite error message:

Line 84: verb not recognized

If you use IBM's BASIC Compiler, you know what a pain it is to locate errors in your source code — errors are referenced to the .LST file, and multiple lines share the same line number. Line numbers advance erratically, and the number referenced in the error message can almost never be directly located — it falls somewhere in a range of perhaps ten lines. Clipper uses your source file for reference (what a novel idea) and simply counts each ASCII line, whether blank or not. Your word processor will tell you exactly what line you're on as you page through your source code.

The 84th line of the source program was:

REPORT FORM PHONES

A quick look in the Clipper manual confirmed that this command was not implemented, but would be in the future. I considered throwing the product out, because it meant that simple reports would have to be coded by hand, but then I realized that I never use the report driver in any "serious" work anyway, and I could probably do without it for a while.

The next peculiarity to surface was the fact that Clipper handles indexing differently than dBASE. Although your original dBASE data files (.DBF) are just fine, the dBASE indexes (.NDX) are not used. Clipper contains a utility to create .NTX files instead. Conversion is painless, assuming you documented what fields were used to create the index in the first place. If you didn't (very naughty), you'll have to go into dBASE, USE the file with the index, and then do a DISPLAY STATUS so that you can see what you had in mind when you created the index.

The next major difference caused the most difficulty. When dBASE sees a "DO" command in your code, it dutifully trundles off to perform that piece of code. In some cases, it runs an entirely new program, and in other cases dBASE merely executes code from a procedure file, after which it returns to what it was doing. When I first used dBASE III, I was very disappointed that procedures couldn't really be used as subroutines — dBASE takes forever to load in the code, and seems to reload it every time it's called. The lack of a proper subroutine in dBASE results in very lengthy, repetitive coding.

Clipper, though, has an entirely different idea of how to treat a "DO" command. It looks for the procedure file from which that code comes, or it looks for the program which will be chained-to from the current program. If it finds the source code in your subdirectory, it simply sucks that code in and compiles it with all the rest! I must admit I wasn't prepared for this, and found it very alarming — until I tried to run the compiled code.

What happens is that a complex system of programs suddenly becomes one large program — and it really flies! Not only does the individual code execute much faster, but no run time is wasted loading code — it's already in your program! Think about the implications!

Of course there is a price to pay for this, and it comes in two forms. First, Clipper, like many compilers, won't accept the sort of ambiguity associated with chaining from a macro. You

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

can't "DO &PROGRAM" because Clipper doesn't know what code to suck in and compile, and it doesn't want to wait until you try to run the program to find out — it's too late then. This means that you may have to rewrite some sections of your favorite programs to explicity branch on a choice and "DO PROG1" in one branch, "DO PROG2" in another, and so on — generally not very elegant when compared to "DO &PROGRAM."

The second penalty is that your program grows very quickly in size. Most people don't care that it takes a few extra seconds at the start to load a very large program if it eliminates all the chaining activity during execution, so this isn't the problem with size. The problem is that as you continue to add modules, each subsequent compile and link takes longer and longer. If you make a change in only one small module, you have to recompile and relink the whole thing. One system of 70 .PRG files that I worked on in Clipper took about 25 minutes to compile and link — I got caught up on a six-months backlog of obscure computer magazines during one week of programming.

Clearly, this is not an acceptable situation, and the folks at Clipper have provided an intelligent alternative. If Clipper can't find the source code referred to by the "DO," it assumes that you've got an object file (.OBJ) of that name lying around somewhere and waits for the linker to put all the missing pieces together. This means, in theory, that you have to recompile only the module you're working on and then link the whole series.

In practice, though, you need to hide your source code in another subdirectory, or off on a floppy somewhere — if Clipper can find it, it'll use it. You also need to plan your modules more carefully, and chart the location of all chained-to programs and called procedures. In a complex system, the tendency is to think that the next compile and link will be your last — that all bugs have been fixed and it'll be easier to just compile and link EVERYTHING rather than go to the trouble of hiding the source code, transferring over each module and compiling it into an object file, removing the source code again, and linking all the object code.

I seem to learn more slowly than most other programmers, and it took me a week to realize that this growing problem had gotten completely out of hand — my next Clipper project will start out broken up and be managed that way from the start. There's nothing inherently difficult about the process, but it may be different from the way you currently write code. Those of us who write code from the top down and stub out the branches will have to remember to comment out the branches and not leave unwritten "DO" commands lying around. Clipper objects.

Although small .PRG files turn into large .EXE files, you can compile a whole system of .PRG files into an .EXE file that's not much larger. The 2,560-byte phone list file turned into 118,928 bytes of .EXE file, but a whole system of seventy .PRG files, taking up almost ten times as much space (210,302 bytes), compiled into an .EXE file only half again as big (174,096 bytes). Part of the dramatic difference, though, was because many of the .PRG files were very small and had the usual remark headers on them. Obviously the compiler throws out all the comments, so heavily commented programs take up proportionally less space.

If the size of the finished product is likely to be a problem, Clipper provides a method for overlaying code. This is convenient if your .EXE file gets too large for the user's memory, or even too large to fit on a single floppy. You'd have to write a pretty good sized program in order to need the overlays, and they do add some complexity to your design and coding chores. Most people probably won't need them.

Nantucket has made a number of enhancements to dBASE III, none of which affected my programming in the least, but which are nonetheless impressive and which might prove crucial for *your* application:

- 64,000 active variables instead of 256
- 1024 fields instead of 128
- 250 files open at once instead of 15; (it actually opens and closes your files as necessary if you have too many open — this is transparent to your coding)
- SET RELATION TO allows one parent and unlimited children instead of only one child
- Has a user defined function
- Can call outside programs, with some restrictions (must be object files and available to the linker)
- Can put memo fields in memory variables and do searches, etc.
- Can create help files for your application that can be invoked by the user by pressing F1 while the program is waiting for an input; you create a separate help file with custom prompts and information
- Has a debugger that can single-step and use breakpoints; can be removed after testing has been completed; lets you dump any variables

A number of commands, or functions, from dBASE III don't work in quite the same way or have been replaced in Clipper. ASSIST, BROWSE, CHANGE, EDIT, INSERT, LIST STRUCTURE, and MODIFY COMMAND are not implemented, ostensibly because they are interactively oriented and largely useless in a compiled environment. JOIN, READ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SAVE, SCIENTIFIC NOTATION, SET PATH, SORT, TOTAL, and UPDATE are supposed to be implemented, but haven't been yet. I didn't miss any of them, but one of these might be crucial to *your* application.

Several commands that are primarily used for debugging, such as DISPLAY MEMORY and DISPLAY STATUS, can be invoked in a different way with the debugger utility. CREATE LABEL, CREATE REPORT, CREATE STRUCTURE, and their MODIFY forms have been replaced with Clipper utilities (but LABEL and REPORT haven't been implemented in the current release — we hope they'll be along Real Soon Now, as Jerry Pournelle says.)

There are other interesting differences. Date fields require legitimate dates — not just blanks. A savvy user can defeat this by exiting the READ with a Control-Q or an <Esc>. Screen editing a bunch of GETs is somewhat restricted — the four directional arrows are supported, along with HOME, CTRL-HOME, END, and CTRL-END to move to each end of a current GET and to move to the first or last GET. Missing are a few handy WordStar-style dBASE commands like CTRL-T and CTRL-Y.

Runtime errors are not a whole lot better than dBASE's efforts. Fortunately, Clipper messages don't show the whole genealogy of programs that led up to the error as dBASE III does (and MENU.PRG begat ENTER.PRG which begat . . .). They show the file name, the line number, and a class and type code, such as "(1,2)." I found the runtime messages not nearly as helpful as the compiler messages, which are very good, but generally, in spite of being pretty terse, the runtime messages get you close enough to figure out what went wrong. They always appear on the top line of the display and give you the opportunity to quit (and write out data if possible), abort (directly to DOS), or ignore them. Many errors result in multiple error messages, and of course there's almost no point in ignoring an error — things just go downhill from there.

Finally, there are some new goodies that really aren't all that important, but which are fun to use. You can define a frame (as in picture frame) by setting up a string with the four corners and four sides of the box you want to create. You then call the BOX command with the top right and bottom left coordinates (but why not the top left and bottom right, which seems more "logical"?) and the frame string that you defined. Clipper will draw the appropriate box for you. When your program runs, if you look carefully, you can see Clipper draw the box in a clockwise fashion, but don't blink — it's fast.

You can use the CALL function with parameters, and Clipper supports FOR/NEXT statements. If you're a function key advocate, you can assign up to 40 function keys with SET FUNCTION <n> TO <exp>. These initialize the 10 function keys with Control, Shift, and Alternate combinations.

You can accept a string parameter from outside your program so that the user (or a batch file) can type your program name followed by a parameter. It's an interesting feature and should make a lot of people happy.

New functions include FIELDNAME(n) which returns the name of the nth field in the currently selected file; INKEY() which works like BASIC's; LASTKEY() which returns the ASCII for the last key pressed (you're supposed to be able to use it to detect how a READ was exited); LASTREC() which returns the number of records in a file; PROCLINE() which returns the source code line number of the current procedure; PROCNAME() which returns the name of the current program or procedure; and REPLICATE() which works like STRING$() in BASIC.

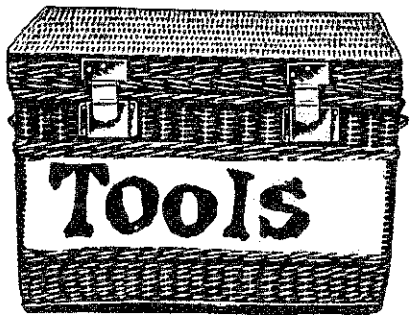
I didn't run any benchmark tests for speed because I didn't think they would prove much. Everyone expects a compiler to beat the pants off an interpreter, and Clipper really flies. It definitely paints the screen faster than IBM compiled BASIC (not meant to damn with faint praise), and Nantucket claims their programs run 2 to 20 times faster than equivalents running under an interpreter. You won't have any trouble seeing the improvement, but I'll leave it to the stopwatch set to create voluminous comparison tables. Benchmarks would probably be more useful in comparing Clipper with another dBASE III compiler, instead of with dBASE III itself.

Nantucket provides an interesting twist on customer support. Although they have phone service, they charge for it. They really want you to access support through their Private Network on The Source. To make this easy, they include a free membership to The Source and will send you a user ID and password, rate information, local Source phone number, and log-on procedures. As this is written, I haven't tried their support, and in fact haven't needed it (wish I could say the same for dBASE). They claim they have information, problem-solving ideas, and assembly language drivers for customer use. The idea seems great — a few dollars an hour for connect time on The Source means that support is relatively inexpensive and should be almost instantaneous if on-line hand-holding is not required. I've found the Programmer's Data Base SIG on CompuServe very helpful for dBASE support from other users, and apparently Ashton-Tate already uses The Source for support, too, so we'll see how this works out.

Depending on your needs and programming habits, you can see that Clipper may offer just what you need, or it may lack a certain crucial function. It fits *my* programming habits very well, and the dramatic increase in speed, lack of need to distribute source code, and lack of need for any other programs by the end user make it an ideal tool for me. Nantucket has done to Ashton-Tate what the sleek Clipper ships did to the old sailing vessels plying the tea trade with China — made it almost obsolete. Until someone invents the equivalent of a programming steamship, Clipper and competitive compilers will probably take most serious development work away from good old — dBASE II and III. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BLAISE'S CTOOLS2



by Rex Jaeschke

Price: CTOOLS2 $100
Blaise Computing Inc.
2034 Blake Street
Berkeley, CA 94704
(415) 540-5441

Volume 3, Number 2 of Programmer's Journal *carried Rex Jaeschke's review of Blaise's CTOOLS, a library of general purpose functions callable from C programs and written with the help of the Microsoft/Lattice compiler (Versions 1.4 and 2.0) and Computer Innovations' C86 compiler (Version 2.01A). In this issue Rex Jaeschke reviews Blaise's sequel, CTOOLS2, a separate library of functions specifically for use with PC/MS-DOS Version 2.x.*

CTOOLS2 is a separate library of functions along the lines of CTOOLS. There is no overlap in functionality between the two libraries. Neither library requires the other (although several of the CTOOLS2 demonstration programs do use routines from CTOOLS). Both libraries can, however, be used by the same program. While CTOOLS is a library of general purpose functions, CTOOLS2 is designed specifically for use with PC/MS-DOS V2.x.

In terms of packaging, documentation, and installation, CTOOLS2 is very similar to CTOOLS. Throughout the rest of this review we'll concentrate on the unique capabilities of the CTOOLS2 routines. As in CTOOLS, all source code is provided with CTOOLS2, and it is of the same high quality.

The library is divided into five functional groups:

1. Support Utilities
2. Program Control and Memory Management
3. File Handling
4. Directory Maintenance
5. DOS Information.

The Support Utilities routines provide access to DOS function calls and include a capability to use the information made available by DOS. The functions give access to the segment register, allow blocks of memory to be moved to and from any location addressable by the processor, allow any register to be initialized prior to a call to DOS, and can invoke any DOS function.

A series of memory management functions can be used to allocate and free memory at run-time and to expand or compress dynamically allocated memory blocks. DOS Version 2.x introduced the Program Environment concept, which allows logical variables to be equated to physical character strings. A function to find the logical-to-physical translation for a named variable is supplied. However, only the first 127 characters of the environment are searched. Programs can load and execute other programs using the DOS EXEC function; program exit codes can be set and tested; and DOS commands can be executed from within a user program. The test program INVOKE that is included with the kit clearly demonstrates these last three capabilities, including running a batch procedure from a program. This set of routines allows a menu-driven DOS shell to be written very easily.

The file handling functions overlap with some of the routines generally provided with a compiler's library. However, this allows a uniform function naming convention and makes the whole file interface consistent throughout CTOOLS2. Facilities are provided to open and close files, set and get file attributes, move the file pointer, read and write to files, delete files, and set and get the Disk Transfer Area.

Several functions are provided to manipulate subdirectories — by creating and removing them, setting and getting the current default, setting and getting the default disk drive, finding file name entries using wild-card naming, renaming files, and setting and getting file creation/update date and time. The DOS system date and time can be set or returned, and the version of DOS can be determined. The amount of free space on a disk can also be found.

Although the above-mentioned capabilities are no doubt useful, the set of routines that pertain to interrupt vectors and their service routines is by far the most exciting part of the library. The documentation section introducing these functions provides some of the best information about interrupts and their service routines as they apply to DOS that is available and, more important, understandable, to the average programmer.

The interrupt-handling functions allow the programmer to write an Interrupt Service Routine (ISR) entirely in C without being concerned with DOS and PC specifics. Briefly — the ISR functions allow you to set and get an interrupt vector, to disable and enable maskable interrupts, and to create an interrupt service dispatching environment with code and stack space that will allow proper handling of multiple ISRs. The DOS capabilities to generate a software interrupt and to terminate but stay resident round out the ISR support group.

Although the documentation's discussion of ISRs will probably be somewhat difficult to comprehend for novice DOS-internals programmers, two excellent demonstration programs (in source code) are provided to illustrate the concept. A CLOCK program defines and installs an ISR that executes

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

every tic of the clock (about 18.2 times each second). The ISR causes a digital clock to appear in the top right corner of the screen. When the ISR is installed, the user may specify the foreground attribute (color) for the clock display, the frequency of the alarm bell, and the optional message to be displayed when the alarm is triggered. Once the ISR has been installed, the program SETCLK can be run to change the clock foreground attribute and alarm interval — by modifying the ISR database defined by program CLOCK.

Note that once an ISR has been defined and loaded, there is no way to remove it under DOS. This can be done only by rebooting. The DOS Version 2.x GRAPHICS program is an example of a DOS-supplied ISR. It traps all printer output generated by the CTRL/PRTSC key combination and handles graphic displays for standard dot-matrix printers. If you have a printer other than that supported by GRAPHICS, you may wish to write your own ISR to do this. Other possible uses for an ISR are in scientific and engineering applications where certain programs need to be triggered by significant events or by timers expiring. ISRs could be triggered by inputs to hardware ports, from an RS-232C channel or some other real-time interface.

CTOOLS2 is a worthy adjunct to CTOOLS, and if you are seriously looking at writing some interrupt handling code and you have doubts about your ability to write it in assembler, or if you don't have the time to reinvent the wheel, you would do well to look at this package. Even if your application has different requirements from an ISR than are provided by CTOOLS2, using the source code provided will significantly reduce your design and development time — and all for the price of about two man-hours. □

Rex Jaeschke is a consultant and Editor of The C Journal. *He regularly writes articles and columns on the C language and other computer-related topics for U.S. and European publications.*

Yes!

Please enter my new subscription to

programmer's journal

Bi-Monthly Subscription Rate

	U.S.	Canada (U.S. Funds only)	Foreign
1 year	☐ $24	☐ $30	☐ $40
2 years	☐ $44	☐ $55	☐ $75
3 years	☐ $64	☐ $80	☐ $110

Mail to: P.O. Box 3000 Department EE
Denville, NJ 07834

☐ New Subscription
☐ Renewal

☐ Check enclosed
☐ Bill me
☐ Charge my Bankcard: ☐ VISA ☐ MC
Card No. _____
Expiration date _____

Signature

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Telephone _____

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WINDOWS AND SCREENS...

The Research Group has released SAYWHAT, a graphic screen and program generator for dBase II and III, Turbo Pascal, and BASIC. SAYWHAT designs data entry screens for Mono, Color, and Mono Graphic adapters and generates the needed code in the language that you're using. Screens are stored on disk. According to the authors, SAYWHAT lets you graphically enter the location of all your dBASE GET statements, (or Turbo ReadLn, or BASIC INPUT statements) and permits you to enter variable names and picture clauses. SAYWHAT will name variables if you don't. SAYWHAT, not copy-protected, is $39.95.

The Research Group
88 South Linden Ave.
South San Francisco, CA 94080
(415) 571-5019

Softway has released a Turbo version of MATIS, which is also available for BASIC, C, Assembler, and Microsoft Pascal. The Turbo version, MATIS/T, is a set of Assembler routines, called as Turbo Pascal procedures, that design and edit data entry screens, develop multiple windows, and allow printing and management of screens from within an application. MATIS/T supports color and mono cards, creates screens up to 65534 lines by 65534 columns, and has auto scroll. It costs $31.95, including shipping.

Softway Inc.
500 Sutter Street
San Francisco, CA 94102

On-Line Help allows you to design help windows, which are stored in condensed form in a single DOS file called a Help Library. Windows are called into your program at a keystroke, On-Line Help restores the screen on exit. The Library Interface is a 1500 byte module that is combined with your program at link time, or in various other ways. On-Line Help is currently callable from Interpreted and Compiled BASIC, Lattice and Microsoft C, Turbo Pascal, and a host of other languages. It costs $149.

Opt-Tech Data Processing
P.O. Box 678
Zephyr Cove
Nevada 89448
(702) 588-3737

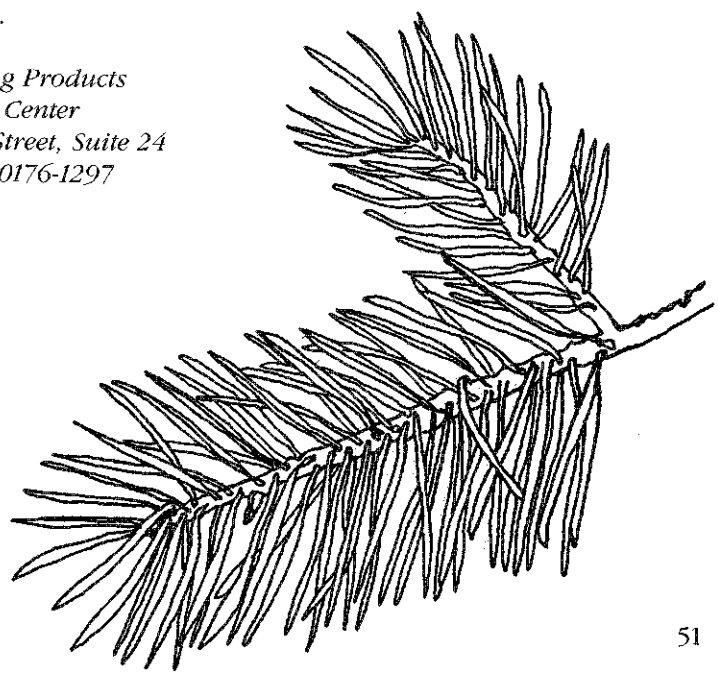
King Programming Products has added KPP-HELP for BASIC to its existing version for dBase II and dBase III. It now works with interpreted BASIC, and King hopes to have a compiled BASIC version out soon. The programmer builds and numbers up to 32,000 screens, specifying the position where each is to appear. The manual guides the programmer through setting up interfaces to the screens in the BASIC program. KPP-HELP sells for $50.

King Programming Products
O'Hare Aerospace Center
4825 North Scott Street, Suite 24
Schiller Park, IL 60176-1297
(312) 671-6218 □

RELIEF FROM SPREADSHEET SEGREGATION

Vertex Systems has released PIPELINE, a utility that translates datafiles and templates from one spreadsheet file format to another. PIPELINE can accept the following types of files: Lotus 1-2-3 (.WKS or formula print), Symphony (.WRK), Visicalc (.VC and .DIF), Supercalc (.SDI or contents report), dBASE II (.DBF), Multiplan (.SLK), or ASCII data (comma-separated variables, etc.) Any of these files can be translated to Lotus 1-2-3, (.WKS), Symphony (.WRK), Visicalc (.VC or .DIF), Supercalc (.SDI), dBASE II (.DBF), Multiplan (.SLK), or ASCII data files. PIPELINE translates text and numerical values, formulas, functions, and screen formatting commands. Vertex's Apple Turnover or Xeno-Copy PLUS offer file transfer between the Apple II, many CP/M computers, and the IBM PC. PIPELINE runs on the IBM PC, XT, AT, and most IBM PC compatibles. Versions for CP/M-80, UNIX, and XENIX are on the way.

Vertex Systems
6022 West Pico
Los Angeles, CA 90035
(213) 938-0857 □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

PJ Billboard *is a product and services directory containing listings for hardware and software of interest to the IBM PC programmer. The goal is to provide our readers with a wide range of information on current products and updates.* ***PJ Billboard*** *serves our advertisers by providing a low cost, flexible format in which they can present their products to our readers.*

The standard layout is four columns, 9 picas wide, with copy set in 9 point on 10 point leading. ***PJ Billboard*** *offers a 400 character product description, including your company name, address, and telephone number. Boldface and italics available.*

The listings are $65, pre-paid, non-commissionable for a one time insertion. Listings must be received typed, preferably on one of our convenient order forms. Copy deadlines are the same as the normal ad materials closing dates, which can be found on our rate card, or you may call the ***Programmer's Journal*** *Advertising Department at 503-484-0602 for more information.*

BASIC Programmers: F*I*F (Fill-In-Formats) saves time! Easily create great looking input screens w/all edit options. A generated compact BASIC source code subroutine, compilable, handles entire input session w/user. One GOSUB does it all. Price $49.95. Visa. PCjr & up, 128K, 80 col. monitor, graphics card (except Portable or Jr).
Siskiyou Software,
830 Garden Way,
Ashland, OR 97520,
503/482-1908

CODE SIFTER finds the bottlenecks in your program. Menu driven, online HELP, define 32 areas automatically or manually, direct symbolic statistics to any device/file, automatic repartition & rerun multiple times. New release 1.20. Demo available. PC/XT/AT, DOS 2.0 or later. Money back guarantee. $119.
David Smith Software
Box 25A RD#3
Oxford, N.Y. 13830
(607) 843-6209

FOOTHILL MARKETING FOR IBM PC MARKETERS
Direct Marketing Services. Updated IBM PC Mail Lists. Direct Response Cards. Co-operative Mailing. Creative Referral.
For information please call Foothill Marketing
408-373-8373

All IBM-PC users interested in Software Tools! *PC TOOLCHEST* is the bi-monthly newsletter for you. Information on the latest software for program development, text processing, graphics, and statistics. Utilities and Pubic Domain software also featured. Classified and Display ads. A 6 issue subscription is only $20.
PC TOOLCHEST,
Box A,
6241 Sunderland Drive,
Columbus, Ohio 43229,
614-846-7069 □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AUTHOR GUIDELINES

Our ad for writers in the July/August 1985, issue of *Programmer's Journal* (Volume 3, Number 4) produced a number of good articles and a lot of inquiries about writing for *PJ*. The most persistent question has been, "What are your formatting requirements?" We've always asked authors to submit their articles on disk with accompanying hard copy. Beyond that, we've had a long-standing policy that our authors shouldn't be burdened with a set of exacting criteria for article submission. Our feeling has been that it's up to us to see that a good article gets into our format and into print.

But because so many of you have asked us what format we prefer, we've come up with a short list that can serve as *PJ*'s author guidelines:

1. Submit the article on disk with an accompanying hard copy to

   Programmer's Journal
   P.O. Box 30160
   Eugene, OR 97403

2. Use the word processor of your choice. One that produces standard ASCII text files is easiest for us to work with (PC Write, Perfect Writer, Microsoft Word, for example). WordStar devotees, don't be discouraged by this preference of ours; we have a handy conversion utility furnished by Michael Hoyt that "cleans up" your files — that is, turns them into ASCII text files and deletes your formatting commands.

3. Don't indent pargraphs. Put two carriage returns between paragraphs. Put your left margin in Column 1.

4. Resist the temptation to use your word processor's formatting commands for boldface, underlining, etc. Indicate the emphases you want in red ink on the hard copy that accompanies your disk.

5. Use subheads to let the reader know what sections of your article are about. Put subheads (and the article title) in upper case.

6. Attach some information about yourself if you'd like it to appear at the end of your article. Let us know if you'd like us to print your address so that readers can get in touch with you.

Expect a response to your article submission within the month. Send us a stamped, self-addressed envelope if you'd like us to return your disk and manuscript.

ADVERTISERS INDEX

Blaise Computing	32
Borland International	Cov. 4
Computer Faire, Inc.	37
Data Base Decisions	42
Datalight	21
Ecosoft, Inc.	16
Essential Software	20
Gimpel Software	44
Greenleaf Software	40
Landmark Software	34
Lifeboat Associates	23
Micromotion	9
Opt-Tech Data Processing	13
Peter Norton Computing	17
Phoenix Computer Products	6
Programmer's Shop	27
Raima, Inc.	31
Rational Systems, Inc.	7
Software Horizons	31
Sterling Castle	43

Patronize our advertisers. Along with you readers, they keep PJ going. When you write or call to ask about their products or to order, let them know you saw their ad in Programmer's Journal.

CORRECTION

The table at the end of Bernard H. Robinson, Jr.'s "DOS 2.0 Sort and Find Filters" (*PJ*, July/August 1985, Volume 3, Number 4) should have appeared at the end of his "Calendar Date Routines in BASIC" (*PJ*, March/April 1985, Volume 3, Number 2).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU AN ISSUE SHY OF A FULL SET?

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILEWNED, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBAse II, Pascal's pointer power.
- Vol. 2, No. 6: Periscope—Building a hardware board for the IBM PC, AT&T surprises all, dBase II to dBASE III?, Intel 80286 bugs by FNE, calendar functions for BASIC.
- Vol. 3, No. 1: November's Comdex, C—its uses and abuses, Using the C preprocessor, Memory segmentation examples, Index files (part 1), Structured programming language, Periscope, a symbolic debugger.
- Vol. 3, No. 2: UNIX! (again), Artificial intelligence, Doctor C's tips — pointers to functions, Calendar date routines in BASIC, Calculate record layouts, automatically!
- Vol. 3, No. 3: NAPLPS and the professional programmer, Instant-C, Printer control from a batch file, Repositioning the stars in Starfinder, Reducing EXE file size through data positioning, tools for better programming.
- Vol 3, No. 4: More UNIX stuff, C programming books, C code excerpts, How to cure the 'dreaded locked keyboard' condition, Calling MS-DOS from Turbo Pascal, DOS 2.X sort and find filters, the SET Tools.
- Vol. 3, No. 5: Windows done here; C programming books, Part Two; C code excerpts, Part Two; A menu-driven PC using DISPLAY.COM; Turbo Pascal and MASM: making them work together; dBASE II routines; Turbo Pascal 3.0; TopView Programmer's Toolkit.

Back Issues are $5.50 ea. PREPAID ONLY

($6.50 Canada, $8.50 foreign, U.S. FUNDS ONLY)

Please send me the following Back Issues:

Vol. I # ___ ___ ___ ___ ___ ___

Vol. II # ___ ___ ___ ___ ___ ___

Vol. III # ___ ___ ___ ___ ___ ___

Bi-Monthly Subscription Rates

☐ New Subscription ☐ Renewal

Term	U.S.	Canada (U.S. funds only please)	Foreign
1 year 6 issues	☐ $24	☐ $30	☐ $40
2 years 12 issues	☐ $44	☐ $55	☐ $75
3 years 18 issues	☐ $64	☐ $80	☐ $110

☐ Check enclosed Total amount _____

☐ Charge my Bankcard: ☐ VISA ☐ MC

_____ Expir. Date _____

Signature

Send to:

PROGRAMMER'S JOURNAL

P.O. Box 30160

Eugene, OR 97403

(503) 484-2162

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SAVE OVER 30% ON OUR GIFT PACKS!
60-DAY MONEY-BACK GUARANTEE

How Borland's Three New Holiday Packs Will Fill Your Stocking Without Emptying Your Piggybank.

Three special packs with dazzling discounts that will help get you into a Holiday mood. You can get some of Turbo, most of Turbo, or all of Turbo—including the two newest members of the Turbo family, Turbo GameWorks™ and Turbo Editor Toolbox.™ You also get our unmatched 60-day money-back guarantee, quality products that aren't copy-protected.

TURBO NEW PACK $95.00.

You get the two exciting new members of the Turbo Pascal family,

- TURBO GAMEWORKS, Chess, Bridge, and Go-Moku, complete with source code and a 200-page manual.
- TURBO EDITOR TOOLBOX, all the building blocks to make your own editors and word processors, complete with source code and a 200-page manual.

TURBO HOLIDAY PACK $125.00.

You get all three of the Turbo family classics for only $125.00 (about a 30% discount). Turbo Pascal 3.0 and Turbo Tutor and Turbo DataBase Toolbox—all for just $125.00.

- TURBO PASCAL combines the fastest Pascal compiler with an integrated development environment.
- TURBO TUTOR teaches you step-by-step how to use Turbo Pascal with commented source code for all program examples on diskette.
- TURBO DATABASE TOOLBOX offers three problem-solving modules for your Turbo Pascal programs: Turbo Access, Turbo Sort, and GINST, which generates a ready-to-run installation program that lets you forget about adapting your software to specific terminals.

TURBO HOLIDAY JUMBO PACK $245.00.

This is it—the whole thing, the entire Turbo family including its two newest members. You get:

- Turbo Pascal
- Turbo Tutor
- Turbo GameWorks NEW!
- Turbo Graphix Toolbox
- Turbo DataBase Toolbox
- Turbo Editor Toolbox NEW!

and you pay only $245.00 for all six! Which means that you're getting everything at only about $40 a piece. Quite a holiday deal. (And if you already own one or several members of the Turbo family, be creative—nothing can stop you from buying the Jumbo Pack, picking out the ones you already have and giving the rest as holiday gifts to family and friends. At these prices you can afford to give to others and to yourself.) Speaking of Holidays, this offer lasts until March 31, 1986. (At Borland, we like to make the Holidays last.)

NEW! **TURBO GAMEWORKS $69.95.**

Our new Turbo GameWorks offers games you can play and replay without Turbo Pascal or revise and rewrite with Turbo Pascal 3.0. We give you the source code, the manual, the diskettes and the competitive edge. Chess, Bridge and Go-Moku. State-of-the-art games that let you be player, referee, and rules committee all at once because you have the Turbo Pascal source code. Learn exactly how the games are made—so you can go off and make your own. And Turbo GameWorks is the only quality game you can buy that is not copy-protected. Sold separately, only $69.95. (Just $47.50 if you buy the Turbo New Pack.)

NEW! **TURBO EDITOR TOOLBOX $69.95.**

Build your own word processor—for only $69.95! You get ready-to-compile source code, a full-featured word processor that looks and acts like WordStar™, and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. With Turbo Editor Toolbox you can have the best of all word processors. You can make WordStar behave like Multi-Mate. Support windows just like Microsoft's Word. And do it as fast as WordPerfect does it. Incorporate your new "hybrids" into your programs to achieve incredible control and power. Sold separately, only $69.95. (If you buy the Turbo New Pack, the price drops to just $47.50.)

NOT COPY-PROTECTED

Holiday Gift Packs, Turbo GameWorks™ & Turbo Editor Toolbox™

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133.

		Quantity
*Turbo Holiday Jumbo Pack	$245.00	
*Turbo Holiday Pack	$125.00	
*Turbo New Pack	$95.00	
Pascal	$69.95	
Pascal w/8087	$109.90	
Pascal w/BCD	$109.90	
Pascal w/8087 and BCD	$124.95	
Turbo DataBase	$54.95	
Turbo Graphix	$54.95	
Turbo Tutor	$34.95	
Turbo Editor	$69.95	
Turbo GameWorks	$69.95	

These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

The holiday packs include an upgrade coupon for both options so you get BCD and 8087 support for $39.95 (regularly $55.00).

*Gift Pack Offers Last Until March 31, 1986

Carefully describe your computer system!

Mine is: ___ 8-bit ___ 16-bit

I use: ___ PC-DOS ___ MS-DOS ___ CP/M-80 ___ CP/M-86

My computer's name and model is: _____

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Name: _____

Shipping Address: _____

City: _____ Zip: _____

State: _____

Telephone: _____

H36

Amount: (CA add 6% tax) _____

Payment: VISA MC Check Bank Draft

Credit Card Expiration Date: ___ / ___

Card # _____

COD's and Purchase Orders will not be accepted by Borland International. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank.

NOTE: Turbo Editor Toolbox and Turbo GameWorks are available for the IBM PC and true-compatibles using Turbo Pascal 3.0 ONLY.

BORLAND INTERNATIONAL
4585 Scotts Valley Drive, Scotts Valley CA 95066
Phone (408) 438-8400 Telex 172373

Copyright 1985 Borland International BI-1017

Turbo Pascal and Turbo Tutor are registered trademarks and Turbo DataBase Toolbox, Turbo Graphix Toolbox, Turbo Editor Toolbox, Turbo GameWorks, and MicroStar are trademarks of Borland International, Inc. WordStar is a trademark of MicroPro International Corp. Multi-Mate is a trademark of Multimate International Corp. Microsoft is a registered trademark and Word is a trademark of Microsoft Corp. WordPerfect is a trademark of Satellite Software International.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED